Semiautomatic Improvements of System-Initiative Spoken Dialog Applications Using Interactive Clustering

Dong Yu, *Member, IEEE,* and Alex Acero, *Fellow, IEEE*

***Abstract*—While many successful spoken dialog systems have been deployed over telephone networks in recent years, the high cost of developing such applications has led to limited adoption. Despite large research efforts in user-initiative and mixed-initiative systems, most commercial applications follow a system initiative approach because they are simpler to design and are found to work adequately. Yet, even designing such system-initiative spoken dialog systems has proven costly when compared with simpler touchtone systems. To address this issue, we describe in this paper our efforts in building diagnostics tools to let nonexperienced speech developers write usable applications without the need for transcribing calls. Our approach consists of two steps. In the first step, we cluster calls based on Question/Answer (QA) states and transitions, analyze the success rates associated with each QA state and transition, and identify the most problematic QA states and transitions based on a criterion we call Arc Cut Gain in Success Rate (ACGSR). In the second step, we cluster calls associated with problematic QA transitions through an approach we term Interactive Clustering (IC). The purpose of this step is to automatically cluster calls that are similar to those already labeled by the developers to maximize productivity. Experiments on an internal auto-attendant application show that our approach can significantly reduce the time and effort needed to identify problems in spoken dialog applications.**

***Index Terms*—Automatic analysis, call transition diagram, data mining, model-based clustering, semi-supervised clustering, speech recognition.**

I. Introduction

INTERACTIVE VOICE response (IVR) systems [6] allow users to call a phone number and interact with a machine. IVR systems are commonly used these days as a front-end to reach a customer service center, a particular department within an enterprise, or to serve as a voice portal, among other uses. Traditionally, IVR systems have used recorded prompts for system output and accepted touchtone keys from the user as input. Over the past decade, we have seen a proliferation of IVR systems that accept not only touchtone keys but the callers' speech as well.

The number of ports an IVR system has indicates how many simultaneous calls it can take. The number of speech ports, which measures the number of telephone channels serviced by an automatic speech recognizer, has undergone a rapid increase in the 1990s. However, growth in speech ports has slowed considerably recently. The main barrier to wider adoption stems from the cost incurred in developing spoken dialog applications [25].

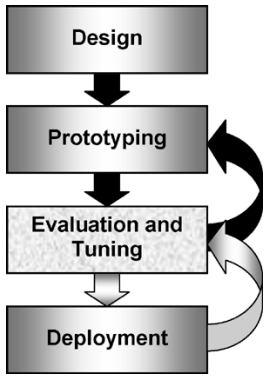

Fig. 1. Typical development cycle of speech applications. The evaluation and tuning phase is usually the most costly part.

A typical spoken dialog application includes application logic, dialogs, grammars, a speech recognition engine, etc. Since it is unlikely that the system performs adequately right at the very beginning, building a quality spoken dialog application usually involves the four steps depicted in Fig. 1, namely, design, prototyping, evaluation and tuning, and deployment. Among all these four steps, evaluation and tuning is one of the most important phases and usually the most costly since it typically involves running the system for several weeks, manually transcribing the recordings, manually analyzing them to determine the system performance, and making changes to the application to get ready for another iteration.

The tuning phase happens both during the test and pilot stages as well as after the deployment, although it is usually more important to tune the system after the system has been deployed since more real data are available at that stage. In the evaluation and tuning phase, developers identify areas that can be improved and adjust the system accordingly. For example, developers can tune the confidence threshold, language model weights, and prompts, in order to boost performance. Developers may want to rephrase a given prompt if they find that many callers do not say anything since this often means the prompt was confusing. Developers may also want to rephrase the prompt to increase the odds that callers will say things within the grammar (e.g., "Who do you want to contact?" versus "Please say the first and last name of the person you want to contact"). Alternatively. they may also want to change the grammar to better cover callers' responses (i.e., even a

Manuscript received August 14, 2004; revised April 18, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Mazin Gilbert.

The authors are with the Speech Research Group, Microsoft Research, Redmond, WA 98052 USA (e-mail: dongyu@microsoft.com; alexac@microsoft.com).

Digital Object Identifier 10.1109/TSA.2005.851876

simple yes/no grammar has been observed to fail miserably in some regions, where callers tend to say "yes ma'm" instead). In the latter case, they may decide to also augment the grammar. The tuning phase can take many months and requires a team of developers, testers, and speech technology experts.

A lot of effort has been put on reducing the total cost needed to develop and deploy spoken dialog applications. For example, Microsoft, Nuance, Scansoft, Philips, and several other companies offer a number of development tools, but we noticed that important functionality is not available in such tools. For instance, spoken dialog application developers usually do not know how to further increase customer satisfaction for their application, even though they have access to a large number of application logs such as those automatically provided by Microsoft Speech Server. It is thus very valuable to have a tool that automatically (or semi-automatically) determines with what callers are struggling or which parts of their application need the most work. This is exactly what we wish to achieve with our approach.

In this paper, we report our recent work in the spoken dialog application problem identification through performance analysis and clustering. We aim to reduce the time and effort needed by developers to identify problems in their system-initiative spoken dialog applications at the evaluation and tuning phase. Despite a large body of research in user initiative and mixed initiative systems [2], most commercial applications follow a system-initiative (also called system-directed) approach because they are simpler to design and are found to work adequately. Yet, even designing such system-initiative spoken dialog systems has proven costly when compared to simpler touchtone systems. To address this issue, we describe in this paper our efforts in diagnostics tools to let nonexperienced speech developers write solid applications without the need to transcribe calls.

We assume that developers have access to the spoken dialog application logs since logging is nowadays a standard facility in most spoken dialog application platforms (such as Microsoft Speech Server). We also assume the dialog is built as a state machine, where each state is a dialog turn or Question/Answer state. Our approach thus helps developers mine and analyze the log data and consists of two steps. In the first step, we cluster calls based on QA states and transitions. We then identify the most problematic QA states and transitions based on a criterion we term Arc Cut Gain in Success Rate (ACGSR.) We devise an algorithm to estimate the ACGSR by analyzing the Call Transition Diagram (CTD) that is automatically inferred from the spoken dialog application log data. In the second step, we cluster calls associated with problematic QA transitions through an approach we call Interactive Clustering (IC). The purpose of IC is to automatically cluster calls that are similar to those labeled by the developers so that developers can focus on analyzing those unlabeled calls and identifying new problems and, thus, maximize productivity. Experiments on an auto-attendant application show that our approaches can significantly reduce the time and effort needed to identify problems in spoken dialog applications.

The rest of the paper is organized as follows: In Section II, we describe the call clustering method based on QA states and transitions using the concept of ACGSR to measure the importance of the QA transitions. In Section III, we describe how to identify problems in the calls associated with those problematic QA transitions using the concept of IC. In Section IV, we report the performance evaluation of our algorithms on our internal auto-attendant application. We show that our approach does significantly reduce the time and effort needed by the developers to identify problems. We present related work in Section V and conclusions in Section VI.

II. Step One: Cluster and Rank QA Transitions

In this section, we describe the call clustering algorithm based on QA states and transitions as well as the ranking algorithm of QA transitions based on ACGSR. We first introduce the concept of CTD and the way to automatically build it from spoken dialog application logs.

The ultimate goal of performance evaluation and tuning is to improve the caller satisfaction. Because this is not an objective measure that we can directly optimize, we will focus instead on maximizing the success rate of the spoken dialog application.

Definition 1: The *success rate* r is defined as the percentage of successful calls S, i.e., calls that fulfilled their tasks, over all calls placed N and is given by $r = S/N$.

Fulfillment of task is defined by the spoken dialog applications and means that the call was successfully transferred to the right person in an auto-attendant application or that the user obtained the desired information in a banking application. For simplicity, we consider a call placed for two tasks as two separate calls. The start and end of a task are determined by the developers of the application and automatically marked in the logs. In this paper, we will use the success/failure markings for an auto-attendant application, but for some applications, defining success/failure may not be so straightforward. It is up to the application developer to add events to the call log that mark such success or failures.

A. Call Transition Diagram and Its Construction

Definition 2: A *CTD* is a 3-tuple CTD $= (Q, A, T)$, where we have the following.

- Q is a finite set of *states* $Q = \{q_i | i = 1, \ldots, I\}$.
- A is a finite set of *arcs* where $A = \{a_{ij}\} \subseteq (Q \times Q)$.
- T is the set of calls passing through the arcs $T = \{T_{ij}\}$, where T_{ij} represents all the calls associated with the arc a_{ij}.

A call c can be represented as a sequence of QA states traversed and the associated parameters. In other words, $c = \{q_{x_k} | k = 1, \ldots, K\}$, where q_{x_1} is the start state, and q_{x_K} is the end state.

Note that the CTD differentiates itself from the traditional Call Flow Diagram (CFD) in that the CTD contains the set of calls passing through each arc. The CTD is also different from the traditional Markov model description of the call states. In the traditional Markov model description, the number of times a state and/or arc is visited determines the model parameters, while in the CTD, we measure the number of unique calls visiting each arc. No matter how many times the same call has visited the same arc, it is counted as one call in the CTD. The reason for this difference is that we are more interested in successful calls rather than the events.

TABLE I
CTD CONSTRUCTION ALGORITHM

```
For each call Cn {
   For each arc sequentially visited by Cn {
      If the start state of the arc is not in the CTD { add it. }
      If the end state of the arc is not in the CTD { add it. }
      If the arc is not in the CTD {add it. }
      If the call Cn is not associated with the arc
         { add it. }
      Else
         { update the timestamp of the call. }
   }
}
```

TABLE II
SAMPLE LOG

Attribute	Value
Call ID	1119
Event ID	1
QA state	Phase1AskQA
Prompt	Thank you for calling Microsoft. Who would you like to contact?
ASR	Dong Yu
Confidence	0.757
DTMF	
Semantics	NameID: 1376401
Grammars	Ask, Repeat, OperatorRequest
Prompt time	5.18 s
ASR time	3.16 s
Barged-in	No
Task Info	Task starts

Let T_{ij} represent the set of calls associated with arc a_{ij}, and let $t_{ij} = |T_{ij}|$ represent the number of calls in the set. Similarly, we use S_{ij} to represent the set of successful calls associated with arc a_{ij}, and $s_{ij} = |S_{ij}|$ to represent the number of successful calls associated with the arc.

The construction of the CTD from the application logs is straightforward. Table I summarizes the algorithm. Here, a log contains the information on QA states visited by a call. A sample QA in a call log provided by Microsoft Speech Server is shown in Table II. Note that during the CTD construction process, calls are grouped together based on the QA states and transitions they have visited.

B. ACGSR

After we construct the CTD, we need to identify those QA states and transitions that need the most attention from developers. An obvious way is to sort the QA states and transitions based on the local unsuccessful rate (the failure rate against all calls visiting the QA state). This approach, however, is not optimal since the improvement in the local success rate does not necessarily translate into the improvement in the overall success rate. For example, it may not be worth paying attention to a QA state with high unsuccessful rate if that QA state is buried down in a voice menu since the total gain might be very limited. Another approach is to sort QA states based on the overall failure rate (the failure rate against all calls placed). This approach may also be suboptimal. In the example shown in Fig. 2, each transition is associated with two numbers: the total number of eventually failed calls that went through the transition and the total number of calls that visited the transition. If we sort the transitions based on the total number of failed calls (or the overall failure rate), a_{25} should be listed first. However, fixing all the problems associated with that transition (so that all calls flowing through that transition originally now go through a_{26}) does not give us the best possible gain in the success rate since transition a_{26} also has a high failure rate. This suggests that a good criterion is critical to the effectiveness of the ranking. Therefore, we propose a criterion named ACGSR.

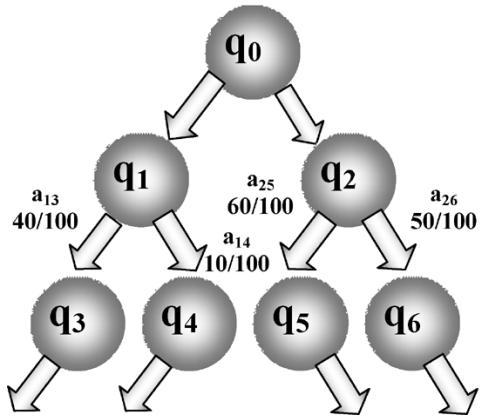

Fig. 2. Example of using overall success rate to sort QA states and transitions. Two numbers are associated with each transition. The first number is the total number of eventually failed calls that went through the transition. The second number is the total number of calls that visited the transition.

Definition 3: The ACGSR is the change of overall success rate if all calls originally passing through the arc were to be distributed to other arcs with the same start state (as if the arc were cut from the CTD), given that all other system parameters are unchanged.

Note that ACGSR measures the *change* of overall success rate when a transition is changed. In other words, it measures the difference of the success rate before and after the transition change. It is not a measurement of the success rate itself, which is available from the logs and CTD directly. There is an ACGSR score for each transition. ACGSR essentially measures how important it is to reduce the number of calls passing through a specific arc. The higher the ACGSR score, the more important it is.

Unfortunately, ACGSR is not directly available from the CTD (or logs) and needs to be estimated. To estimate ACGSR, we assume that calls redirected to other arcs have the same success rate as the calls originally passing through those arcs. Although it does not hold strictly, this assumption is valid most of the time. Because when a call passes through a transition, it is the dialog construction under that transition that determines whether the call will fail. We will show the result of a two-sample proportion hypothesis test to support this in Section IV. If the CTD is a tree with no loop and no merge, as depicted in Fig. 3, the ACGSR of arc a_{ij} can be estimated using

$$\text{ACGSR}(a_{ij}) = \frac{1}{N}\left(\frac{\sum_{k\neq j} s_{ik}}{\sum_{k\neq j} t_{ik}} \bullet t_{ij} - s_{ij}\right) \tag{1}$$

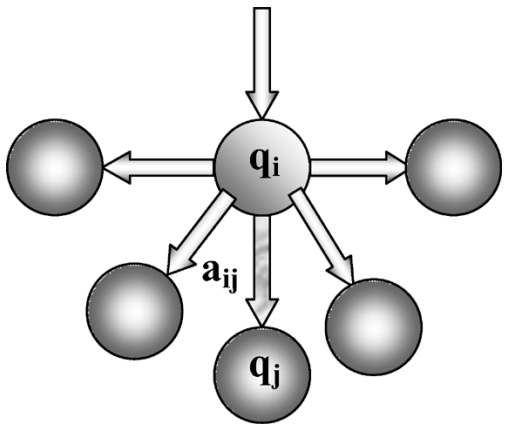

Fig. 3. Estimation of ACGSR in a tree-type CTD. Calls originally passing through the arc a_{ij} are now redirected to other arcs, whose start state is q_i.

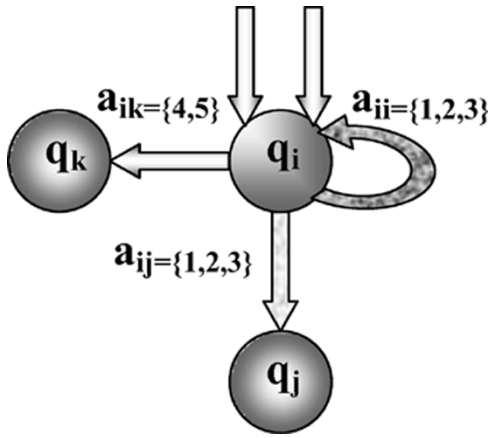

Fig. 4. Estimation of ACGSR in a CTD with loops and merges. In this example, calls 1, 2, and 3 failed, and calls 4 and 5 were successful. The success rate does not change if arc a_{ii} is cut since all calls passing through a_{ii} have already been counted in arcs a_{ij} and a_{ik}. However, if arc a_{ij} is cut, we have a big gain in the success rate.

where k is the destination of an arc. Let us define t_i as the total number of calls passing through state q_i and s_i as the total number of successful calls passing through state q_i, i.e.,

$$t_i = \sum_k t_{ik} \tag{2}$$

$$s_i = \sum_k s_{ik}. \tag{3}$$

Equation (1) thus becomes

$$\begin{aligned} \text{ACGSR}(a_{ij}) &= \frac{1}{N}\left(\frac{s_i - s_{ij}}{t_i - t_{ij}} \bullet t_{ij} - s_{ij}\right) \\ &= \frac{(s_i \bullet t_{ij} - t_i \bullet s_{ij})}{N(t_i - t_{ij})}. \end{aligned} \tag{4}$$

When the CTD is not a tree, however, (4) no longer holds due to the fact that the same call may flow out of a state several times through different arcs. Let us use the states and arcs in Fig. 4 as an example. Assume that calls 1, 2, and 3 are failed and that calls 4 and 5 are successful. It is obvious that the success rate does not change if arc a_{ii} is cut since all calls passing through a_{ii} have already been counted in arcs a_{ij} and a_{ik}. However, if arc a_{ij} is cut, all calls originally failed are now turned into successful calls, and we have a big gain in the success rate.

The example illustrated in Fig. 4 suggests that only the calls flowing out of the start state (no looping back) through the arc should be considered when one estimates ACGSR.

Definition 4: A call *loops back* to the start state q_i of arc a_{ij} from the end state q_j if there is a list of arcs $\{a_{x_k y_k} | k = 1, \ldots, K\}$ such that $x_0 = j$, $x_{k+1} = y_k$, and $y_K = i$, and the call passes through them sequentially.

Definition 5: A *sunken call* of arc a_{ij} is a call that eventually passes through arc a_{ij} and does not loop back to the start state q_i again.

Theorem 1: Call c passing through the nonstop state q_i is a sunken call of one and only one arc whose start state is q_i.

Proof: Since the state is a nonstop state, the call must go out of state q_i and end at another state. This means that c is a sunken call of at least one arc whose start state is q_i. For the same reason, the call cannot be a sunken call of multiple arcs with the same start state.

Definition 6: A *timestamp* is a number associated with each event. Later events have a higher timestamp than the earlier ones. A timestamp can be real time or any number (e.g., event ID) that increases over time. We use $time(c, a_{ij})$ to indicate the timestamps of the event that call c passes through arc a_{ij}. Since each call may pass through the same arc several times, $\max(time(c, a_{ij}))$ is used to indicate the latest time c passes through a_{ij}.

Theorem 2: Call c associated with arc a_{ij} is a sunken call of a_{ij} if and only if $\max(time(c, a_{ij})) > \max(time(c, a_{ki}))\forall k$.

Proof: $\Rightarrow$

If call c is a sunken call, the call eventually flows out through arc a_{ij} and does not loop back again. This means that later events (with a higher timestamp) associated with call c do not occur in all arcs whose end state is q_i.

$\Leftarrow$

We prove this with a counter example. If call c is not a sunken call, there is a list of arcs $\{a_{x_k y_k} | k = 1, \ldots, K\}$ such that $x_0 = j$, $x_{k+1} = y_k$, and $y_K = i$, and the call passes through them sequentially. This means that

$$\begin{aligned} \max\left(time(c, a_{ij})\right) &< time\left(c, a_{jy_1}\right) \\ &< time\left(c, a_{y_1 y_2}\right) \\ &< \ldots \\ &< time\left(c, a_{y_K i}\right) \\ &\leq \max\left(time\left(c, a_{y_K i}\right)\right). \end{aligned} \tag{5}$$

The final algorithm of ACGSR estimation can be obtained by redefining T_{ij} as the set of sunken calls of arc a_{ij} in (5).

C. QA Ranking

To present the results to the developers, we sort arcs on the estimated ACGSR in descending order. Besides this, we need to consider the following two issues.

First, developers are interested in knowing how to *improve* the system. For this reason, those arcs with negative ACGSR are not informative to them and should not be displayed since a negative

ACGSR indicates that reducing the calls flowing through the arc will reduce the success rate.

Second, developers might not be able to reduce the number of calls passing through some arcs, even though those arcs may have large ACGSR values. For example, an application may have a menu from which to select the callers. Transitions to any state associated with an item in the menu should not be cut. To solve this problem, developers can either mark these arcs in the CTD before running the tool or select to hide the transition in the displayed report.

The ACGSR estimation algorithm is useful not only in determining the most important QA states and/or transitions for developers to work on but in many other areas as well. For example, by changing the grammar used in a QA state, some calls may be diverged to a different path than the original one. The ACGSR estimation algorithm can be used to estimate the total success rate gain after the grammar change without running the updated application.

III. Step Two: Identify QA Problems Through Interactive Clustering

After the most problematic QA states and/or transitions are identified, it is desirable to group calls passing through these arcs into clusters based on the cause of problems such as semantic coverage, grammar overgeneralization, channel echo, incorrect barge-in, bad prompt, and programming errors. Ideally, the whole process should be automatic. However, several difficulties exist in doing so. First, it is difficult to know all possible causes of the problems and, therefore, train classifiers for them before hand. Second, different applications may log different information, and it is the additional information that separates one type of problem from another. Third, each application has its own dialog flow, and therefore, it is difficult to generate a set of universal classifiers.

For this reason, we tackle the problem from a different angle and aim to reduce the time and effort needed for developers to identify problems through IC.

Definition 7: IC is a clustering process based on semi-supervised clustering. During initialization, IC clusters calls based on prior knowledge such as pretrained classifiers. If no prior knowledge is available, IC keeps all calls in one cluster or groups them with unsupervised clustering. The developer listens to unlabeled calls and labels them (with the cause of the problem, for example). The developer's interaction is used as supervision and/or feedback for IC to adjust and label other calls. This process goes on until all calls are labeled and the developer is satisfied with the labels.

The essence of the IC is similar to co-training, which combines a small amount of seed-labeled data with an unlimited amount of unlabeled data to bootstrap a classifier [8], [18].

With a brute force approach, the developer needs to check all the calls associated with a special QA transition to identify most of the problems. During this process, the developer is usually frustrated since he/she is listening to calls with the same causes again and again. Traditionally, the calls are sent for manual transcription, but this is costly and results in a significant delay. A slightly better approach is random sampling, but it is hard to determine the number of samples needed. A large number of samples means lots of redundant work, and a small number of samples means a low probability of finding all the problems.

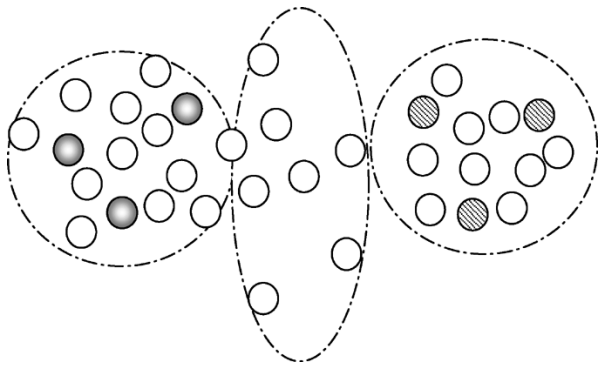

Fig. 5. Example of IC. After six calls are labeled, calls are clustered into three classes, two of which are labeled. The developer needs to check only the unlabeled cluster to find other problems.

With IC, things are different. After the developer labels a call, all similar calls are grouped together and labeled. The developer needs to focus on only those unlabeled calls that are usually associated with new causes. The number of calls the developer needs to listen to overall is thus greatly reduced.

Fig. 5 illustrates an example of IC. Originally, all calls are unlabeled and clustered into one class, as shown in Fig. 5. After only six calls are manually labeled, calls are clustered into three classes, two of which are labeled. The next call the developer needs to listen to is one from the unlabeled class.

The main component of IC is a semi-supervised, finite mixture distribution model based on clustering algorithm.

A. Membership Determination

We want to find the best classification in which instances similar to labeled data are labeled accordingly, given a large set of unlabeled data instances and a small set of labeled data instances. For the sake of easy discussion, we denote a variable by a token in upper case (e.g., X_i) and the value of the variable by that same token in lower case (e.g., x_i). We use $p(x|y)$ to denote the probability that $X = x$, given $Y = y$, or a probability distribution for X, given Y.

Our approach to cluster the whole data set $X = \{x_i | i = 1, \ldots, I\}$ into J clusters $C = \{c_j | j = 1, \ldots, J\}$ is based on the mixture distribution model.

Given the model parameters λ, the probability that instance x_i belongs to class c_j is

$$p(c_j|x_i, \lambda) = \frac{p(c_j|\lambda)p(x_i|c_j, \lambda)}{\sum_{j'=1}^{J} p(c_{j'}|\lambda)p(x_i|c_{j'}, \lambda)} \tag{6}$$

where $p(x_i|c_j, \lambda)$ is the probability that instance x_i is generated by class c_j and $p(c_j|\lambda)$ is the prior of different classes. We assume that the prior follows a Bernoulli distribution, i.e., without knowing the value of the data instance, the probability that it belongs to class c_j is

$$p(c_j|\lambda) = \pi_j. \tag{7}$$

This means (6) can be expressed as

$$p(c_j|x_i,\lambda)=\frac{\pi_j p(x_i|c_j,\lambda)}{\sum_{j'=1}^{J}\pi_{j'}(x_i|c_{j'},\lambda)}. \quad (8)$$

The probabilities $p(c_j|x_i,\lambda)$ are called membership probabilities. After we have computed these probabilities, we can either assign the data instance to the cluster with the highest probability (a hard assignment) or assign the data instance fractionally to the set of clusters according to this distribution (a soft assignment). When we present the result to the developers, we use hard assignment.

B. Model Parameter Learning

The model parameters are estimated with the Expectation–Maximization (EM) algorithm [14]. The EM algorithm starts with a set of initial values of the parameters and then iterates between an expectation or E step and an maximization or M step until the parameter values converge to stable values.

Given the model parameters λ, the probability that instance x_i is generated by the model is specified as

$$p(x_i|\lambda)=\sum_{j=1}^{J}\pi_j p(x_i|c_j,\lambda). \quad (9)$$

The probability that all data instances are generated by the model is

$$p(X|\lambda)=\prod_{i=1}^{I}p(x_i|\lambda)=\prod_{i=1}^{I}\sum_{j=1}^{J}\pi_j p(x_i|c_j,\lambda). \quad (10)$$

The posterior probability of model parameter λ is

$$p(\lambda|X)=\frac{p(X|\lambda)p(\lambda)}{p(X)} \quad (11)$$

where $p(\lambda)$ is the prior distribution of λ. The maximum *a posteriori* (MAP) parameter is defined as

$$\begin{aligned}\lambda^{\mathrm{MAP}}&=\arg\max_{\lambda}p(\lambda|X)\\&=\arg\max_{\lambda}p(X|\lambda)p(\lambda)\\&=\arg\max_{\lambda}\left(\log p(X|\lambda)+\log p(\lambda)\right).\end{aligned} \quad (12)$$

When used in conjunction with vague or noninformative priors, MAP estimates are smoothed versions of Maximum Likelihood (ML) estimates [20]. We choose a Dirichlet distribution as the prior for the multinomial distribution with parameters $\phi=(\phi_1,\ldots,\phi_N)$

$$p(\phi_1,\ldots,\phi_N|\alpha_1,\ldots,\alpha_N)=\frac{\Gamma\left(\sum_{n=1}^{N}\alpha_n\right)}{\prod_{n=1}^{N}\Gamma(\alpha_n)}\prod_{n=1}^{N}\phi_n^{\alpha_n-1}. \quad (13)$$

In our case, we choose $\alpha_1=\ldots=\alpha_N=1/N$.

Given the current parameters λ, the new parameters λ' are thus estimated to maximize the following function:

$$\begin{aligned}Q(\lambda',\lambda)&=E\left(p(\lambda'|X)\right)\\&=\sum_{i=1}^{I}\sum_{j=1}^{J}p(c_j|x_i,\lambda)\left[\log p(X|\lambda')+\log p(\lambda')\right].\end{aligned} \quad (14)$$

Assume that each data instance is an ordered vector of K attribute values $x_i=\{x_{ik}|k=1,\ldots K\}$ and that attributes are independent of each other; therefore

$$p(x_i|c_j,\lambda)=\prod_{k=1}^{K}p(x_{ik}|c_j,\lambda). \quad (15)$$

Each attribute can have either nominal values or real values. Multinomial value attributes are modeled with a Bernoulli distribution: $\lambda_{jk}=\{q_{jk1},\ldots,q_{jkL_k}\}$. Real value attributes are modeled with a Gaussian distribution: $\lambda_{jk}=\{\mu_{jk},\sigma_{jk}\}$.

By maximizing the Q function with respect to each subset of parameters, we can get the algorithms for updating parameters.

At the initialization step, random probabilities are assigned to $p(c_j|x_i,\lambda)$, which is used to estimate the initial model.

C. Supervision as Constraint and Feedback

Thus far, we have only discussed the clustering algorithm without any labeled data. To use the data instances labeled by developers, we artificially boost the importance of the data instance to the estimation of the model parameters

$$p(c_j|x_i,\lambda)=B\bullet\delta\left(j,Lab(i)\right) \quad (16)$$

where $Lab(i)$ is the label of data point x_i, and B is the boost factor. B is originally set to 1 and doubles its value every time the converged result does not have labeled data correctly classified. It is easy to see that the labeled data is used both as constraint [30] and as feedback [11] in this approach.

Since we are interested in helping developers focus on calls that are not similar to those already labeled, we set the number of clusters in the clustering algorithm as one plus the number of known classes. In other words, J is set to 1 initially and increases by one each time a new class is discovered by the developers. If the number of instances in the unlabeled cluster is zero after the clustering, IC will confirm it with a second try. IC stops when it confirms that the number of instances in the unlabeled cluster is zero, and the developer is satisfied with the label.

IV. Experiment and Evaluation

We have developed a graphical user interface (GUI) tool using our approach and evaluated it on an internal application named MS Connect, which is an auto-attendant application built on the Microsoft Speech Server. The system allows callers to speak the name of the person they wish to contact, and it provides them with a variety of messaging options. Although we evaluated our approach with an auto-attendant application, we hope this approach can be applied to other system-initiative spoken dialog applications.

The MS Connect service has 33 QA states, 137 transitions, and 14 different grammars and serves to transfer calls to more

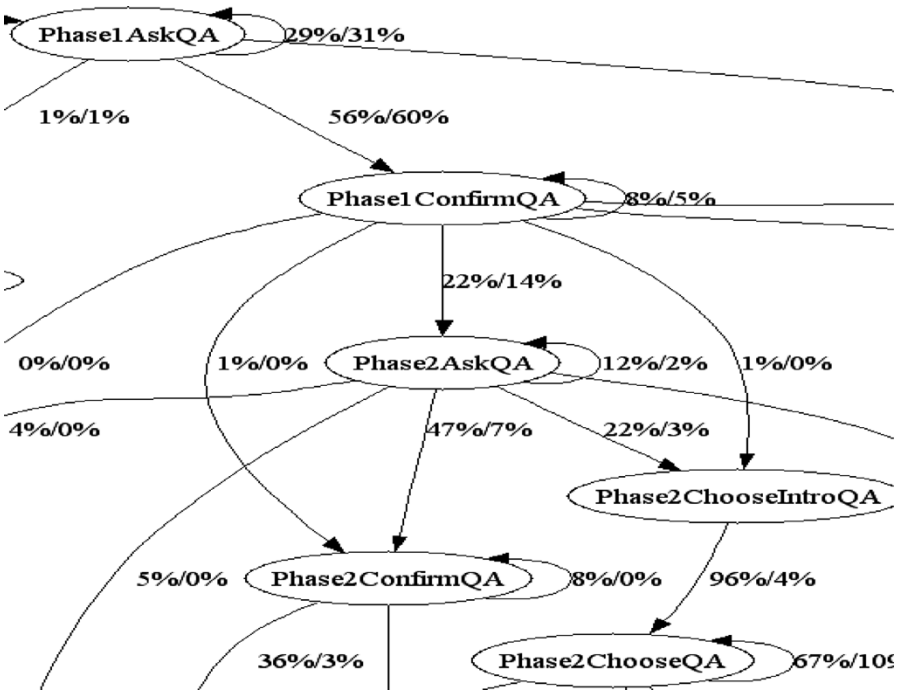

Fig. 6. Part of MS Connect Call Flow Graph.

than 50 000 employees at Microsoft Corporation. Fig. 6 illustrates part of the Call Flow Graph.

For performance monitoring and tuning purposes, all calls are logged through the logging facility provided by Microsoft Speech Server until the call is either transferred or the user hangs up. In our experiment, we used log data collected in June 2004. The whole database contains logs for about 50 000 calls. Each call consists of a sequence of QA events. Each QA event contains information including the QA state name, prompt, automatic speech recognizer (ASR) result, retained audio, confidence score, barge-in information, grammars, Dual-Tone-Multi-Frequency (DTMF) result, prompt time, ASR time, and task information. An example of the event can be found in Table II. The average number of turns for a call is 3.8. The success rate of all the calls is 76.2%, and the average duration of all the calls is 37.3 sec.

Fig. 7 is a snapshot of the GUI tool. It shows the QA transitions and calls sorted based on ACGSR on the June 28, 2004 logs, which contain a total of 2216 calls. From this figure, we can see that the transition that would benefit most from improvement is the transition from Phase1AskQA to FarEnd (hung up by the caller). More than 10% (or 222) calls pass through this transition. If we can reduce the calls flowing through this transition completely, we can get absolute 8.5% (or 189 calls) improvement in the success rate. Unfortunately, this transition means that users hang up after they find they are talking to a machine, so probably, there is not much the developer can do. Another interesting finding is that 1.5% calls pass from SayCallingContact to NearEnd (hung up by the system.) This is surprising since the system already knew where to route the call but failed to do so. It was then discovered that the phone number was not available in the database for some people. Fixing problems in this transition would give us 1.5% gain in success rate. We have run the tools on three days, 10 days, and 30 days of logs and seen the similar results. ACGSR clearly identified those QA states and transitions to which developers should pay most attention.

To check whether the assumption we have made to estimate ACGSR is valid, we compared the success rate associated with transitions before and after changing a transition. For example, we changed the grammar associated with the Phase1AskQA state by manually adjusting pronunciations for hundreds of employee names since the pronunciations generated by the letter-to-sound rules may deviate from the true pronunciations of some names. Table III shows the test result. In the table, n-b represents the total number of calls before the change. S-b represents the success rate before the change. N-a and s-a represent the total number of calls and the success rate after the change, respectively. Note that here, we measure the calls passing through all the transitions started from a special QA state, except for the transition that has been changed. This change reduced the percentage of calls going from Phase1AskQA to FarEnd. Before the change, the system got a consistently low confidence scores for such names as Yun-Cheng Ju and tended to ask for the full name again and again, and many users simply hung up. After the change, we reduced the number of people who hung up before going to the next state. In this case, we measure all the calls passing from Phase1AskQA, except for those passed to the FarEnd state. The variance shown in the table is within the valid deviation range based on the two-sample proportion test. We cannot deny the hypothesis that the success rate before and after changing a transition are the same.

Evaluating the effectiveness of IC requires additional work. Traditionally, clustering algorithms are evaluated with criteria such as confusion matrix, precision, recall, F1 measure, balancing, purity, entropy, and mutual information [15]. These criteria do not fit here since the purpose of IC is to identify all

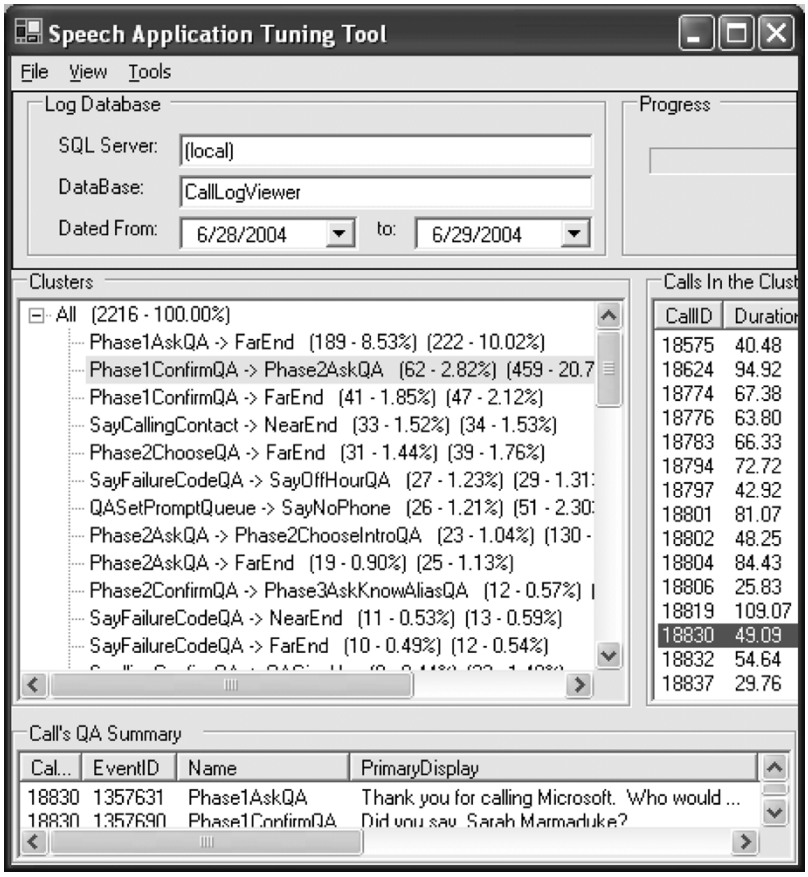

Fig. 7. Snapshot of the GUI tool. It shows the QA transitions sorted by ACGSR. Each transition is followed by two sets of numbers. The first set of numbers indicates the absolute and percentage gain of ACGSR. The second set of numbers indicates the absolute and percentage of calls passing through the arc.

TABLE III
COMPARING THE SUCCESS RATE BEFORE AND AFTER THE CHANGE

Transition From	n-b	s-b	n-a	s-a
Phase1AskQA	1921	0.862	1929	0.860
Phase1ConfirmQA	1788	0.859	1798	0.854

n-b: total number of calls before the change
s-b: success rate before the change
n-a: total number of calls after the change
s-a: success rate after the change

problems as fast as possible. The following are two obvious criteria.

Criterion 1: PPI is the Percentage of Problems Identified over all problems existing under the assumption that the developer can label each call he/she has listened to. The best PPI possible is 100%.

Criterion 2: The *Average Number of Calls Labeled* (ANCL) is the ratio of total number of calls labeled manually per problem identified. The lowest possible ANCL is 1.

Another less obvious criterion is the following.

Criterion 3: The *Problem Distribution Accuracy* (PDA) is the accuracy of the number of calls in each class. It is defined as the Kullback–Leibler (KL) distance [24] between the problem distribution obtained from IC and the true problem distribution

$$\text{PDA} = \sum_i \hat{p}_i \log\left(\frac{\hat{p}_i}{p_i}\right) \tag{17}$$

where $\hat{p}_i$ is the estimated percentage of calls in class i, and p_i is the true percentage of calls in class i. The lowest possible PDA is 0.

PDA is partially related to the goal of the analysis. The closer to 0 the PDA is, the more accurate the estimation of the problem distribution, and the more reliable the developers can count on the estimated problem distribution to prioritize their efforts. PDA by its own, however, is not sufficient to measure the effectiveness of the approach.

We evaluated IC using two sets of data generated from the ACGSR ranking and clustering step. The first set of data contains 459 failed calls manually classified into four types of problems with distributions 338, 36, 53, and 32. The second set of data contains 896 failed calls in four clusters, each of which has 706, 69, 67, and 54 calls. The four types of problems are the following:

- Acoustic confusion, e.g., Alan Meier versus Alan Maier; confusion caused by noise;
- Uncovered semantics, e.g., callers try to reach receptionist or technical support that are not handled by the system;
- Bad dialog design, e.g., callers do not know how to or do not want to choose from a voice menu; callers do not know they need to say the first name and last name;
- Pronunciation variation, e.g., callers' pronunciation of a foreign name is different from that of the ASR.

The four types of problems were identified manually by listening to all calls. The feature set we used in this experiment

TABLE IV
IC PERFORMANCE EVALUATION (DATASET 1)

Approach	Exp Cfg	PPI	ANL	PDA	Acc
Brute force		100%	115	0.00	100%
Sampling	Size=10	71.8%	3.9	0.165	n/a
	PPI=100%	100%	6.62	0.080	n/a
IC	No Cfm	94.5%	1.62	0.045	75.8%
	With Cfm	100%	2.52	0.007	81.1%

TABLE V
IC PERFORMANCE EVALUATION (DATASET 2)

Approach	Exp Cfg	PPI	ANL	PDA	Acc
Brute force		100%	224	0.000	100%
Sampling	Size=10	61.3%	4.73	0.163	n/a
	PPI=99.5%	99.5%	6.99	0.076	n/a
IC	No Cfm	92.5%	1.72	0.032	74.1%
	With Cfm	99.5%	2.55	0.004	79.4%

includes promptID, prompt barged-in status, DtmfID, repetition of passing, duration, and engine confidence scores. We ran 100 independent experiments for each data set and compared IC with full sampling and random sampling. The goal of the experiment is to determine how many problems can be found when the developer listens to only a small subset of all those failed calls (based on the success/failure marking in the log), and hopefully, which call belongs to which problem cluster. Tables IV and V and Fig. 8 illustrate the results. To tie part of the evaluation into the traditional framework, we also report the average precision (Acc) of the classification, although it is not directly related to the purpose of problem identification.

Full sampling guarantees the correct identification of all problems and distributions. However, it requires the developer to listen to 115 calls per problem on average in dataset 1 and 224 calls per problem in dataset 2. The random sampling approach gives us a better result. To compare apples to apples with IC, we conducted two sampling configurations. The first approach is to fix the sampling size, and the second is to fix the PPI. The result shows that by having the developer listen to only ten calls, IC with confirmation can detect close to 100% of the problems in both datasets, whereas the random sampling approach can detect only 71.8% of problems in dataset 1 and 61.3% of problems in dataset 2. To detect close to 100% of problems, IC requires checking only 2.5 calls per problem in both datasets, whereas the random sampling approach requires checking 6.6 calls per problem in dataset 1 and 7.0 calls per problem in dataset 2. In both configurations, IC provides a more accurate distribution than the random sampling approach, whereas IC also clusters similar calls.

Fig. 8 shows the distribution of the number of problems identified if the developer listens to only ten calls. In the random sampling approach, we may identify all four problems with about a 20% of chance in dataset 1 and with less than 10% chance in dataset 2. With the IC approach we can identify all four problems almost all the time. This indicates that developers can identify more problems with our approach if they listen to a fixed number of calls. It also gives us a rough idea of the relationship between the number of calls labeled and the number of classes numbered.

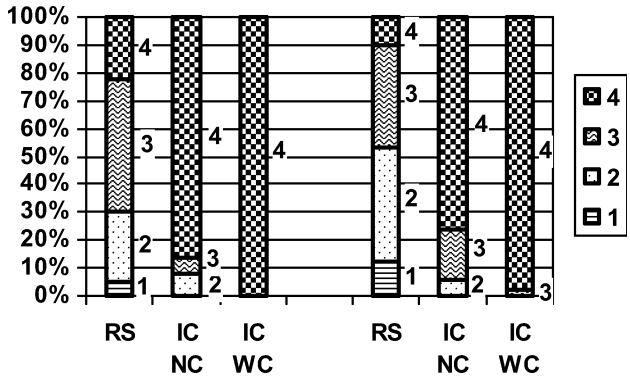

Fig. 8. Distribution of the number of problems identified when the developer listens to only ten calls. The first three bars compare the results for dataset 1, and the last three bars compare the results for dataset 2. RS=Random Sampling. IC NC=IC with No Confirm. IC WC=IC with Confirm. We can see that with IC WC, the developer can find all the problems while devoting little time to it.

TABLE VI
PROBABILITY THAT EACH PROBLEM MAY BE FOUND OUT BY LISTENING TO ONLY 10 CALLS FROM DATASET 1

Approach	Exp Cfg	P1	P2	P3	P4
Brute force		100%	100%	100%	100%
Sampling	Size=10	100%	56%	72%	52%
IC	No Cfm	100%	95%	100%	90%
	With Cfm	100%	100%	100%	100%

TABLE VII
PROBABILITY THAT EACH PROBLEM MAY BE FOUND OUT BY LISTENING TO ONLY TEN CALLS FROM DATASET 2

Approach	Exp Cfg	P1	P2	P3	P4
Brute force		100%	100%	100%	100%
Sampling	Size=10	100%	55%	54%	46%
IC	No Cfm	100%	95%	94%	85%
	With Cfm	100%	100%	100%	98%

Tables VI and VII report the probabilities that each individual problem may be found out by listening to only ten calls from datasets 1 and 2, respectively. We can see that problem 1 can be identified with a 100% chance using either approach. However, we only have about a 50% of chance to find problems 2 and 4 using the random sampling approach, although we can identify those problems with more than an 85% chance using interactive clustering with no confirmation and over a 98% chance using interactive clustering with confirmation.

V. RELATED WORK

The CFD has been widely used in speech dialog design [1]. The CTD is an extension to CFD.

Lots of work has been conducted in dialog-spoken application analysis. Hastie *et al.* [21] reported training a problematic dialog identifier to classify problematic human–computer dialogs. Litman *et al.* [26] and Hirschberg *et al.* [22] suggested adapting the system automatically by detecting ASR misrecognitions. Bechet *et al.* [5] proposed applying data clustering methods to spoken-dialog corpora and then using the resulting clusters for system evaluation and language modeling. Aberdeen *et al.* [1] proposed automatically identifying errors by using semantics attached to the dialogs. Although we also work on spoken dialog

application analysis, our focus is on reducing the time and effort needed to identify problematic QA states and transitions as well as associated problems through automatic and semi-automatic clustering algorithms. In contrast to other approaches, our approach does not require completely labeled training data. Instead, the system learns from the interaction with the developers gradually.

Probabilistic model-based clustering is well known and widely used [3], [5], [9], [13], [17], [23], [27], [29], [32]. Kamvar *et al.* [21] proved that pair-wise distance/similarity-based approaches can be considered to be special cases of model-based clustering.

According to its original definition, clustering is unsupervised. Clusters are solely determined by the initial clusters, distance measurement, unlabeled data samples, and the algorithm used to assign data samples. Recently, however, semi-supervised clustering has been studied rigorously [4], [7], [8], [10]–[12], [16], [19], [28], [30], [31], [33]. In semi-supervised clustering, both labeled and unlabeled data are used to control the clustering process. Labeled data have been used in the following three different ways.

- As initial seeds [4]. The labeled data is used to generate the initial clusters only.
- As constraints [30]. The grouping of labeled data is kept unchanged throughout the clustering process.
- As feedback [11]. Clusters are first determined based on unsupervised clustering iteration, and the labeled data are then used to adjust the resulting clusters iteratively.

Basu *et al.* [4] reported that the constrained approach performs at least as well as the seeded approach. Zhong [33] concluded that "the constrained approach is superior when the available labels are complete while the feedback-based approach should be selected if the labels are incomplete."

Our IC algorithm belongs to the semi-supervised clustering algorithm family, where both labeled and unlabeled data are used in the clustering process. Despite the similarity, our IC algorithm is different from other approaches in several aspects. First, the goal of our approach is to help developers identify problems as fast as possible, whereas in other approaches, the accuracy of the clustering is the most important criteria. For this reason, we introduced PPI, ANCL, and PDA as new criteria. Second, our approach uses labeled data as both the constraints and feedback during the clustering process, whereas other approaches usually use the labeled data as either initial seeds, constraints, or feedback but not combinations of these. Third, in other approaches, the number of clusters is usually assumed to be *a priori* knowledge or determined by criteria such as mutual information. In IC, the number of clusters is determined by the problems already identified by the developers.

VI. Conclusions and Future Work

We described our recent technologies on log analysis and its application to system-initiative spoken dialog application problem identification. In particular, we proposed using ACGSR as the criterion to rank QA state and transitions and using IC to help developers find problems more quickly. We showed that our techniques can significantly reduce the time and effort needed by developers to identify problems in their spoken dialog applications.

We perceive that our system can be extended in the following areas, and we are working in these new areas.

First, when a QA transition is identified to have a large ACGSR, the cause of the problem may not be located at the start state of the transition. For example, calls passing from Phase1ConfirmQA to Phase2AskQA are largely caused by the recognition error in the previous QA state—Phase1AskQA. In our current implementation, we identify those so-called confirmation states and trace back one level to extract a feature set for problem identification purposes. We believe that a better probabilistic approach can be invented to replace this heuristic.

Second, we used only six features in the IC process in the current implementation. However, many other features, even though they are hard to obtain, could be useful for identifying problems. We are working on algorithms to extract those new features.

Third, we do not have prebuilt classifiers for finding common problems in the current implementation. We are working on providing such classifiers once we have enough logs from different applications.

Acknowledgment

The authors thank the speech server team of the Microsoft Corporation for providing us the MS Connect log data. They also thank the members of the speech research group at Microsoft Research, Redmond, WA, and the anonymous reviewers for valuable discussions, comments, and suggestions.

References

[1] J. Aberdeen, C. Doran, L. Damianos, S. Bayer, and L. Hirschman, "Finding errors automatically in semantically tagged dialogues," in *Proc. Human Language Technol.*, 2001, pp. 173–178.

[2] J. Allen, C. I. Guinn, and E. Horvtz, "Mixed-initiative interaction," *IEEE Intell. Syst. Applic.*, vol. 14, no. 5, pp. 14–23, Sep./Oct. 1999.

[3] J. D. Banfield and A. E. Raftery, "Model-based Gaussian and non-Gaussian clustering," *Biometrics*, vol. 49, pp. 803–821, 1993.

[4] S. Basu, A. Banerjee, and R. Mooney, "Semisupervised clustering by seeding," in *Proc. 19th Int. Conf. Machine Learning*, Sydney, Australia, 2002, pp. 19–26.

[5] F. Bechet, G. Riccardi, and D. Hakkani-Tur, "Mining spoken dialogue corpora for system evaluation and modeling," in *Proc. Conf. Empirical Methods in Natural Language Process.*, Barcelona, Spain, 2004, pp. 134–141.

[6] C. Bennett, A. Font Llitjos, S. Shriver, A. Rudnicky, and A. Black, "Building VoiceXML-based applications," in *Proc. Int. Conf. Spoken Language Process.*, Denver, CO, 2002, pp. 2245–2248.

[7] A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph mincuts," in *Proc. 18th Int. Conf. Machine Learning*, 2001, pp. 19–26.

[8] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. 11th Annual Conf. Comput. Learning Theory*, 1998, pp. 92–100.

[9] I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Model-based clustering and visualization of navigation patterns on a web site," *Data Min. Knowl. Discov.*, vol. 7, no. 4, pp. 399–424, 2003.

[10] V. Castelli and T. M. Cover, "The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 2102–2117, Nov. 1996.

[11] D. Cohn, R. Caruana, and A. McCallum, "Semi-Supervised Clustering With User Feedback," Cornell Univ., Ithaca, NY, Tech. Rep. TR2003-1892, 2003.

[12] F. G. Cozman, I. Cohen, and M. C. Cirelo, "Semisupervised learning of mixture models," in *Proc. 20th Int. Conf. Machine Learning*, 2003, pp. 106–113.

[13] P. Cheeseman and J. Stutz, "Bayesian classification (AutoClass): Theory and results," in *Advances in Knowledge Discovery and Data Mining*, U. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Menlo Park, CA: AAAI, 1995, pp. 153–180.
[14] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc.*, vol. B, no. 39, pp. 1–38, 1977.
[15] B. Dom, "An Information-Theoretic External Cluster Validity Measure," IBM, Tech. Rep. RJ10219, San Jose, CA, 2001.
[16] A. Dong and B. Bhanu, "A new semi-supervised EM algorithm for image retrieval," in *Proc. IEEE Int. Conf. Comput. Vision Pattern Recognition*, 2003, pp. 662–667.
[17] C. Fraley and A. Raftery, "How many clusters? Which clustering method? Answers via model-based cluster analysis," *Comput. J.*, vol. 41, pp. 578–588, 1998.
[18] S. Goldman and Y. Zhou, "Enhancing supervised learning with unlabeled data," in *Proc. 17th Int. Conf. Machine Learning*, Stanford, CA, Jun. 29–Jul. 2 2000, pp. 327–334.
[19] A. Guerrero-Curieses and J. Cid-Sueiro, "An entropy minimization principle for semi-supervised terrain classification," in *Proc. IEEE Int. Conf. Image Process.*, 2000, pp. 312–315.
[20] I. Good, *The Estimation of Probabilities*. Cambridge, MA: MIT Press, 1965.
[21] H. W. Hastie, R. Prasad, and M. Walker, "What's the trouble: Automatically identifying problematic dialogs in DARPA communicator dialog systems," in *Proc. Meeting Assoc. Computational Linguistics*, 2002, pp. 384–391.
[22] J. B. Hirschberg, D. J. Litman, and M. Swerts, "Generalizing prosodic prediction of speech recognition errors," in *Proc. 6th Int. Conf. Spoken Language Process.*, 2000, pp. 254–257.
[23] S. Kamvar, D. Klein, and C. Manning, "Interpreting and extending classical agglomerative clustering algorithms using a model-based approach," in *Proc. 19th Int. Conf. Machine Learning*, 2002, pp. 283–290.
[24] S. Kullback and R. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, pp. 79–86, 1951.
[25] A. Kozminski, "System architectures for speech—A practical guide to lowering costs," in *Proc. AVIOS Speech Developers Conf. Expo.*, Mar. 31–Apr. 3 2003.
[26] D. J. Litman, M. A. Walker, and M. J. Kearns, "Automatic detection of poor speech recognition at the dialogue level," in *Proc. 37th Annu. Meet. Assoc. Computational Linguistics*, 1999, pp. 309–316.
[27] G. McLachlan and K. Basford, *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker, 1988.
[28] K. Nigam, A. Mccallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Machine Learning*, vol. 39, pp. 103–134, 2000.
[29] P. Smyth, "Probabilistic model-based clustering of multivariate and sequential data," in *Proc. of Seventh Int. Workshop Artificial Intelligence Statistics*, San Francsico, CA, 1999, pp. 299–304.
[30] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl, "Constrained k-means clustering with background knowledge," in *Proc. 18th Int. Conf. Machine Learning*, 2001, pp. 577–584.
[31] H.-J. Zeng, X.-H. Wang, Z. Chen, H. Lu, and W.-Y. Ma, "Cbc: Clustering based text classification requiring minimal labeled data," in *Proc. IEEE Int. Conf. Data Mining*, Melbourne, FL, 2003, pp. 443–450.
[32] S. Zhong, "Probabilistic Model-Based Clustering of Complex Data," Doctoral dissertation, Dept. Elect. Comput. Eng., Univ. Texas, Austin, TX, 2003.
[33] ——, "Semi-Supervised Model-Based Document Clustering: A Comparative Study," Tech. Rep., Dept. Comput. Sci. Eng., Florida Atlantic Univ., Boca Raton, FL, 2004.

Dong Yu (M'97) received the B.S. degree in electrical engineering from Zhejiang University, Zhejiang, China, the M.S. degree in electrical engineering from the Chinese Academy of Sciences, Beijing, China, and the M.S. degree in computer science from Indiana University, Bloomington.

He joined Microsoft in 1998. His research interests include speech processing, pattern recognition, and network security. He has published more than 20 papers and applied for more than ten patents in these areas.

Mr. Yu has served as a reviewer for many conferences and journals including ICASSP, Interspeech, and the *Journal of Computer Security.*

Alex Acero (S'85–M'90–SM'00–F'04) received the Master's degree from the Polytechnic University of Madrid, Madrid, Spain, in 1985, another Master's degree from Rice University, Houston, TX, in 1987, and the Ph.D. degree from Carnegie Mellon University, Pitsburgh, PA, in 1990, all in electrical engineering.

He was with Apple Computer's Advanced Technology Group, Madrid, from 1990 to1991. In 1992, he joined Telefonica I+D, Madrid, as manager of the speech technology group. In 1994, he joined Microsoft Research, Redmond, WA, where he became Senior Researcher in 1996 and manager of the speech research group in 2000. Since 2005, he has been Research Area Manager overseeing speech, natural language, communication, and collaboration. He is currently an affiliate Professor of electrical engineering at University of Washington, Seattle. He is author of the books *Acoustical and Environmental Robustness in Automatic Speech Recognition* (Boston, MA: Kluwer, 1993) and *Spoken Language Processing* (Englewood Clifs, NJ: Prentice-Hall, 2001), has written invited chapters in three edited books and over 100 technical papers, and has given keynotes, tutorials, and other invited lectures worldwide. He holds ten U.S.patents. His research interests include speech recognition, synthesis and enhancement, speech denoising, language modeling, spoken language systems, statistical methods and machine learning, multimedia signal processing, and multimodal human-computer interaction.

Dr. Acero served on the Speech Technical Committee of the IEEE Signal Processing Society between 1996 and 2002, chairing the committee from 2000 to 2002. He was Publications Chair of the ICASSP Conference in 1998, Sponsorship Chair of the 1999 IEEE Workshop on Automatic Speech Recognition and Understanding, and General Co-Chair of the 2001 IEEE Workshop on Automatic Speech Recognition and Understanding. He has served as Associate Editor for the IEEE SIGNAL PROCESSING LETTERSand is presently Associate Editor for *Computer Speech and Language* and the IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING.