Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)

Toward Open-Set Text-Independent Speaker Identification in Tactical Communications

Matt B. Wolf, WonKyung Park, Jae C. Oh, Misty K. Blowers
Syracuse University, NY 13244, USA

Abstract—**We present the design and implementation of an open-set text-independent speaker identification system using genetic Learning Classifier Systems (LCS). We examine the use of this system in a real-number problem domain, where there is strong interest in its application to tactical communications. We investigate different encoding methods for representing real-number knowledge and study the efficacy of each method for speaker identification. We also identify several difficulties in solving the speaker identification problems with LCS and introduce new approaches to resolve the difficulties. Experimental results show that our system successfully learns 200 voice features at accuracies of 90% to 100% and 15,000 features to more than 80% for the closed-set problem, which is considered a strong result in the speaker identification community. The open-set capability is also comparable to existing numeric-based methods.**

Keywords—**classifier systems, genetic algorithms, language and speech, machine learning**

I. Introduction

This paper discusses the design and implementation of an adaptive speaker identification system based on genetic learning classifier systems (LCS). Speaker identification (SID) can be categorized into *text-dependent* and *text-independent* problems. In text-independent speaker identification, the system should identify the speakers regardless of the words spoken, while text-dependent identification requires the speakers to say predetermined words in order for the person to be identified. We can also categorize the SID problem as *closed-set* or *open-set*. In a closed-set SID system, the system decides the identification of speakers from within a fixed set of known speakers. In fact, the system force-identifies the input voice profile to the closest speaker known to the system. Unlike the closed-set speaker identification problem, the open-set SID problem does not assume the number of speakers is fixed. An open-set SID system can add new speaker profiles dynamically.

Open-set speaker identification is much harder than the closed-set problem because it is difficult to decide whether to introduce a new profile for a speaker or to identify the input as one of the existing speakers. Unfortunately, the common approaches, including many statistical clustering methods, are not suitable for the open-set problem because of their inflexibility. Furthermore, both closed-set and open-set systems must be robust to noise and changes in voices of speakers, for example due to illness. Many statistical methods are not flexible enough to adapt to such changes dynamically after deployment.

LCS [16], [28], [12], [24] are open-ended adaptive learning systems that can learn new rules for changing environments. This adaptability motivated us to use LCS for the speaker identification problem. For the LCS model, we have chosen the XCS classifier system [7] since XCS provides several improvements to the traditional LCS and it is more recently studied and updated. To the best of our knowledge, our system is the first LCS-based speaker identification system. We refer to our system as Speaker Identification-Learning Classifier System (SID-LCS).

This research was partially funded by an AFSOR grant, award number: FA8750-04-1-0259, through ARFL/IFEC, Rome, NY and by an NSF/ITR MEDIUM grant, award number: 0121658.

In order to design a LCS for the SID problem, we need to represent voice features for the LCS in an appropriate classifier format. We have studied several different encoding methods including hyperrectangular, hyperspheroidal, and general hyperellipsoidal methods [10], [7] for feature representation. Another challenge in using XCS for speaker identification is the sheer scope of the input data space. Existing XCS theoretical papers [7], [27] have thus far analyzed problem domains with three dimensions, population sizes less than 6400, and unknown precision real numbers. The task set before our modified system encompasses 14 dimensions, population sizes reaching more than half a million, and at least six digits of precision. We have developed new techniques for SID-LCS to improve the speaker identification capability and address this complexity. Also presented is our algorithm for open-set discrimination, intended for tactical communications [14].

Experimental results show that our system successfully learns 200 human voice features extracted from four different speakers at accuracies of 90% to 100%. It can also learn 15,750 feature vectors from 45 speakers at accuracies passing 80%. These can be considered strong results in the speaker identification community [3]. Our system shows promising results in open-set testing.

II. Organization of the Paper

This paper is organized as follows. Section IV presents previous research on the speaker identification problem and learning classifier systems. Section V describes the voice feature vectors and the SID-LCS system developed. This section also describes several encoding methods for voice features and the open-set decision algorithm. Furthermore, we discuss effective levels of applying genetic algorithms in SID-LCS and discuss issues about using LCS for the complex SID problem space. In Section VI, we present experimental results showing the performance of SID-LCS in both closed-set and open-set problem domains. Section VII presents conclusions and future research.

III. Contributions of the Paper

This paper presents the feasibility of using learning classifier systems for the SID problem. The main contributions are:

1. It is the first attempt to solve the SID problem using LCS.
2. It introduces and invites the LCS community to the SID problem.
3. It identifies the complexity of the SID problem using LCS.

4. It investigates the strength and weakness of the LCS system within the domain of the SID problem.
5. It shows the open-set capability of LCS and introduces the *voting method* for improving the identification capability.
6. It examines the performance potential of altering input parameters and reducing classifier complexity.

IV. Background

In this section, we briefly review feature extraction methods for voice signal processing and a traditional speaker identification method. We also introduce the learning classifier systems and discuss XCS [8], an implementation of LCS and its extension, XCSR, for real-valued parameters.

A. Voice Feature Extraction and Statistical SID Algorithm (LBG-VQ)

Most speech processing concepts and systems are based on vocal parameterization and pattern matching or classification. The biological speech production mechanism is transformed to a mathematical representation, known as the Linear Predictive Coding (LPC) model. LPC is the foundation of the vocal parameterization. The basic intent of LPC model is to represent the vocal tract as an all-pole, time-varying filter. When the biological mechanism is considered in this way it allows for the simplification of the human speech production mechanism into an engineering schematic, capturing the biological mechanism in which the air is thrust from the lungs, passing the vocal cords, through the pharynx, and then out the mouth and nasal cavity. With each person, these physical attributes are unique to that individual [15].

All speech production, beginning with a cognitive process, ends with an acoustical waveform. Once the acoustical waveform is derived, it is carried through transmission media, which could include atmospheric propagation (in air), transducer interfaces, and analog or digital delivery systems. When a sample of audio data is collected, the computer characterizes it as an audio waveform. The audio waveform can be translated into a physical state of the vocal tract through LPC [15].

The parameters representing the vocal tract information can be used to derive the LPC filter coefficients. Parameter sets can be derived from the LPC model, such as the LP Cepstrum, which give us the cepstral feature vectors [22].

The real cepstrum is defined as the inverse Fourier transform of the logarithm of the magnitude Fourier transform. The cepstral coefficients can be derived both from the filter bank and linear predictive analyses described above. Filter-bank analysis represents the signal spectrum by the log-energies at the output of a filter-bank, where the filters are overlapping band-pass filters spread along the frequency axis [21]. This representation gives a rough approximation of the signal spectral shape while smoothing out the harmonic structure. Cepstral coefficients have rather different dynamics - the higher coefficients show the smallest variances [21]. SID-LCS is able to account for the smaller variances in order to reduce the search space.

Once the cepstral feature vectors were generated from each audio sample, traditional methods as in [18], [11], [4], [25], [23] then involved the generation of *codebooks* for each speaker. Popular techniques to generate codebooks for each speaker are based upon the LBG_VQ (Linde-Buzo-Gray Vector Quantization) algorithm. The LBG_VQ algorithm clusters input vectors from *training vector* files, and finds 128 centroids. The operation of the algorithm is to continuously reassign feature vectors to clusters and update cluster centroids until no further reassignment is necessary. Codebooks contain 128 codewords; each codeword is a 14-dimensional vector of 32-bit floats.

Once the speaker was characterized in this way, the speaker could be identified from that closed set in the future. However, if the speaker was not a member of the predefined set, the system forced a decision, and resulted in an erroneous prediction. Also, the choice of 128 as the number of codewords per speaker was arbitrary. We continue to examine methods for the system to choose this number based on the number of speakers and input vectors.

The codebooks are compared against *test vector* files, which are in the same format as the training vector files. Each speaker has its own file of test vectors. They contain 4 seconds of audio data that has been converted into cepstral vectors. The short time length of the testing files is consistent with the short utterances available from tactical communications [14].

B. The Learning Classifier Systems

Learning classifier systems [17], [8], [24], [28], [1] are capable of learning and adaptation through the combinations of ideas from rule-based systems, reinforcement learning, evolutionary computing, and other heuristics. The system attempts to find appropriate *state* and *action* sets through numerical rewards from trial and error and by using genetic algorithms.

Usually, the steady state genetic algorithm (GA) is employed by LCS and operates over the whole rule-set at each iteration. In general, the GA uses *roulette wheel selection* to determine which rules will be the parent rules depending on their fitness. Those selected parent rules produce offspring via mutation and crossover in the usual way. The offspring replace existing rules. Replacement targets are often chosen based on the fitness values they received so far [5]. There are many variations from this traditional LCS. Readers are referred to [5] and [13] for more information about LCS.

B.1 XCS

XCS [2], [8] improves traditional LCS in several ways. When input is introduced to XCS, the system tries to match the input to its population. If no match exists, covering is activated. Default covering generates new classifiers that match with the input for each possible action, not just the desired action of the input. This covering procedure with a *prediction value* makes it possible to generate a complete mapping from the problem domain to the set of actions. Unlike the traditional LCS, the fitness of classifiers in XCS is not based on the payoff received but on the accuracy of predictions. In other words, it makes a “best guess” prediction based on the payoff expected for each possible action. The prediction error ϵ estimates the mean absolute deviation of the prediction. Prediction Array P forms to contain the payoff prediction values of each possible action. For details of fitness update in XCS, see [7].

The system next forms an action set A depending on the sum of prediction values in the match set. Another difference is that the GA takes place in this action set instead of the population [8]. In this way, XCS extends the traditional LCS to reinforcement learning. It also provides a way for solving complex problems which have many possible state-action mappings [5]. A detailed description of XCS can be found in [8].

B.2 XCS with continuous real-valued inputs

Since many real-world problems are not easily expressed in ternary representation, there have been attempts to represent problems from the real number domain [27] using XCS. Recently, Butz [7] discussed a real-valued XCS system called XCSR. XCSR is different from XCS in the input interface, mutation operator, and in the details of covering [27]. XCSR also uses an encoding method that is more suitable to real-number problem domains.

B.3 Various encoding methods for real-valued input

Several ways of representing conditions are available in XCS. First, there are center-spread encoding and lower-upper bound encoding that form hyperrectangular conditions in the problem space by setting the boundaries for each dimension. The difference between center-spread encoding and lower-upper bound encoding lies in their way of representing the boundary. They result in a similar condition structure but require slightly different forms of mutation and crossover. For details, see [6]. Recently, Butz [7] introduced new kernel-based spherical and ellipsoidal encoding methods.

The condition parts for the encoding methods are as follows:
1. $C_{lu} = ((l_1, u_1), (l_2, u_2), ..., (l_n, u_n))$
2. $C_{cs} = ((c_1, s_1), (c_2, s_2), ..., (c_n, s_n))$
3. $C_s = (m_1, m_2, ..., m_n, \sigma)$
4. $C_e = (m_1, m_2, ..., m_n, \sigma_1, \sigma_2, ..., \sigma_n)$
5. $C_g = (m_1, m_2, ..., m_n, \sigma_{1,1}, \sigma_{1,2}, ..., \sigma_{n,n)}$

Fig. 1 shows three examples of a hyperellipsoidal condition in the 3-dimensional problem space. It forms more general condition structures than the hyperspheroidal encoding. The following lists the matching conditions for each of the encoding methods:

f_i = value of ith dimension of feature vector, where $i \in I$

1. $M_{lu} = \begin{cases} true & \text{if } l_i \le f_i \le u_i, \forall i \in I \\ false & \text{otherwise} \end{cases}$

2. $M_{cs} = \begin{cases} true & \text{if } c_i - s_i \le f_i \le c_i + s_i, \forall i \in I \\ false & \text{otherwise} \end{cases}$

3. $M_s = \begin{cases} true & \text{if } \theta_s \le cl.ac \\ false & \text{otherwise} \end{cases}$

where $cl.ac = \exp\left(\frac{\|f_i - m_i\|^2}{2\sigma^2}\right)$

4. $M_e = \begin{cases} true & \text{if } \theta_e \le cl.ac \\ false & \text{otherwise} \end{cases}$

where $cl.ac = \exp - \sum\left(\frac{\|f_i - m_i\|^2}{2\sigma_i^2}\right)$

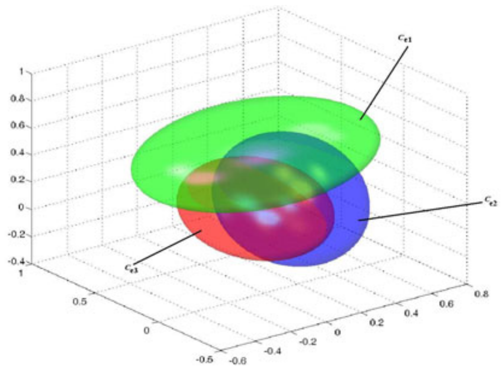

Fig. 1. Illustration of two point crossover of hyperellipsoid in 3-dimensional space.
Topmost: $C_{e1} = (0.1, 0.2, 0.5, 0.4, 0.2, 0.1)$
Lower Right: $C_{e2} = (0.3, 0.2, 0.1, 0.1, 0.2, 0.2)$
Lower Left: $C_{e3} = (0.1, 0.2, 0.1, 0.1, 0.2, 0.1)$

5. $M_g = \begin{cases} true & \text{if } \theta_g \le cl.ac \\ false & \text{otherwise} \end{cases}$

where $cl.ac = \exp - \left(\frac{\sum_{i=1}^{n}(\sum_{j=1}^{n}((f_j - m_j)\sigma_{ij}^2))}{2}\right)$

For crossover, the crossover point can be anywhere in the condition-action part. Mutation happens with certain probability in the condition part. Fig. 1 shows a simple example of crossovers in 3-dimensional space for hyperellipsoid. The crossovers are applied to the topmost and lower-right conditions. The lower-left condition is the offspring which has m_1, m_2 and σ_3 from the topmost condition and the rest of them from the lower-right condition. For details of mutation and crossover in hyperspherical, hyperellipsoidal and general hyperellipsoidal conditions, see [7].

V. The Speaker Identification Learning Classifier System (SID-LCS)

In this section, we introduce our new SID-LCS derived from the XCS. SID-LCS has been implemented based on XCSJava 1.0 by Butz. We will discuss the SID problem domain, proper encoding methods for voice features of 14-dimensional real numbers in section V-A and several issues in the SID-LCS learning in section V-B. The new voting mechanism for better performance in the testing phase is described in section V-E. We discuss a decision algorithm for open-set SID in section V-D.

A. Feature Vectors

Voice feature extraction is done by the LPC model explained in Section IV-A. The speaker's voice is sampled for 4 seconds and this data goes through the LPC algorithm explained above. The result is a list of approximately 250 14-dimensional vectors of real numbers. Fig. 2 shows 240 feature vectors extracted from 4 seconds of voice for one speaker. Each feature vector $F = (f_1, f_2, ..., f_{14})$ is presented to SID-LCS's input interface and matched against the condition part of the classifiers in the current population. If dimensional dependencies are unknown, as in this case, it is advantageous to apply more flexible condition structures as discussed in [7]. We employed hyperrectangles using lower-upper bound, hyperspheroids, and general

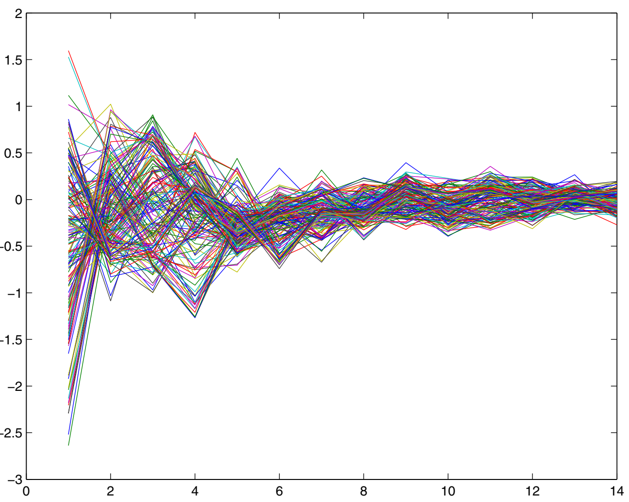

Fig. 2. feature vectors extracted from 4 seconds of female voice. x axis represent dimensions and each line is a single feature vector.

hyperellipsoids for SID-LCS.

B. Complexity of the SID-LCS problem

We have attempted to answer the question of exactly how many feature vectors are sufficient to accurately classify an arbitrarily large number of human voices. This is important for the in-set/out-set case as well, because there must be an underlying base of accurately-classed, in-set speaker vectors to compare against. When considering tactical communications, there are timeliness and data availability factors in real-world SID that limit how many input vectors the system will be able to see before needing to make a decision. The input vectors take the form of a list of 14 real numbers, with at least 6 digits of precision each - a search space of at least $(10^6)^{14}$ number strings (10 digits, 6 places, 14 sets). The system must develop rules able to decide how similar two input vectors can be while still being classified properly.

For a given set of speakers, there will be a few different cases for each input vector from each speaker:

1. When the population of classifiers for a set of speakers is initially empty, the base case will simply add a matching classifier for the input vector to the population through a covering call.
2. When the next input vector for a given speaker is similar enough to an existing classifier for that speaker, it will not need covering since the match set already contains the existing, similar classifier. The new input thus gets "absorbed" by the classifier for a previously covered vector. This is different from subsumption because it is not subject to the experience and boolean tests used for subsumption determination. The "absorption" rate will depend on the encoding method and covering range - larger ranges result in more absorption and thus smaller (potentially less accurate) populations. Since all the hyperspace encoding methods employ some form of real-to-boundary mapping, this absorption will affect all of them to some degree. Potentially useful information for distinguishing two speakers can be lost in this way if the covering range is not fine enough to create separate classifiers. The match set will also contain the classifiers for other speakers which happen to be similar to the current input, causing confusion for the system when the match set is too large.
3. When the next input vector is distinct enough (as determined by the covering range or boundary of existing classifiers), it will be added to the population as a new classifier, taking up another small section of the "*coverable space*" available.

C. Generalization/Specialization

Generalization means to recognize environmental situations having equivalent consequences, but to do so using internal structure of significantly less complexity than the raw environmental data [2]. Thus, generalization can help the system to be robust to noisy situations. The input space in the SID problem, as shown in Fig. 2, is large but it has some consequences that permit generalization. Kovacs' *Optimality hypothesis* [29] suggests that XCS can develop a complete, accurate and maximally compact solution for a given problem. XCS has subsumption procedures to further assist generalization. One of them is *GA subsumption* which checks an offspring classifier to see if its condition is logically subsumed by the condition of an accurate and sufficiently experienced parent. If so, the offspring is not added to the population, but the parent's numerosity is incremented. The other subsumption procedure is *action set subsumption* which takes place in every action set. To be a subsumer, a classifier must first be sufficiently accurate and sufficiently experienced. This can be adjusted by changing ϵ_0, θ_{sub} where ϵ_0 is an error threshold and θ_{sub} is a threshold for experience. In SID-LCS, we have selectively applied both GA subsumption and action set subsumption depending on the experiment.

However, we also need a proper amount of specialization for the open-set environment. In multi-class problems such as the SID problem, generalization over different speakers can be harmful because the system attempts to find the most general and smallest classifier set that can cover all classes altogether. Therefore, even for the open-set problem, we need to generate classifiers that are specified between different speakers in the in-set. In this way, we may evolve classifiers that are maximally generalized and at the same time, specialized enough for different speakers.

D. The Open-set Decision Algorithm

We have introduced an algorithm for making the open-set decision for SID-LCS. As discussed in section IV-B.1, XCS tends to provide complete mappings between state-action. Therefore, for the open-set problem, the system either has no classifiers for the new input from out-set speaker or has some classifiers with low prediction values. The system makes the open-set decision based on this information (see [7] for details of prediction values). Let S be a set of actions, a_i, such that $S = \{a_i | P_{a_i} > \theta_A\}$, where θ_A is a threshold value for in-set speaker prediction values, P_{a_i} is the prediction value for action a_i, and a_i is an action, i.e., speaker i's identifier, where i in *in-set*. The system makes the in-set/out-set decision, i.e., $A = \{a_i, \textit{out-set}\}$ as follows.

$$A = \begin{cases} argmax_{a_i}(P_{a_i}) & \text{if } S \neq \emptyset \\ \text{out-set} & \text{otherwise} \end{cases} \quad (1)$$

Thus, the range of A is from 0 (out-set) to number of in-set speakers. To make this decision the system is trained in the same way as the closed-set training.

E. Voting method

Once the system is trained with a sufficient number of vectors and to a desired accuracy, the actual testing can start. Since we get hundreds of vectors from only 4 seconds of voice data, it is reasonable to let the system decide based on multiple vectors. We can also assume that a certain number of consecutive vectors are from the same speaker. Since the system has been trained with each single feature vector, it can still make a decision based on a single vector first. When the system makes a certain number of decisions, it starts the voting process. If the winner from this voting has enough votes, the decision is finalized. Otherwise, system asks for more vectors until it reaches the available limit.

$$F_v = \begin{cases} \textit{false} & \text{if } win[i] > \theta_{vw} \\ \textit{true} & \text{otherwise} \end{cases} \quad (2)$$

If F_v is true, the system can keep introducing more votes to reach a more accurate decision. So far, we have only used simple majority voting with a threshold θ_{vw} (minimum vote) for an open-set decision.

VI. Experiments

We have used the TIMIT voice data from DARPA [26] for training and testing our SID-LCS system. Our TIMIT data subset provides features from 50 male voices and 50 female voices, generated by the feature extraction method explained in Section IV-A. More than 1000 feature vectors are available for each speaker .

Unless mentioned, parameters were set as follows: $\alpha = 0.1, \beta = 0.2, \delta = 0.1, \nu = 5.0, \theta_{GA} = 25, \epsilon_0 = 10, \theta_{del} = 20, pX = 0.1, pM = 0.1, P_{dontcare} = 0.2, \theta_{sub} = 20, r_0 = 0.3, \theta_r = 0.85, \theta_e = 0.7, \theta_g = 0.45$. These parameters were chosen based on results from previous experiments and consideration of [7].

N specifies the maximal number of microclassifiers in the population. α is the rate of distinction between accurate and nonaccurate classifiers, β is the learning rate, δ specifies the fraction of the mean fitness in the population under which the fitness of a single classifier is considered in the deletion method, ν used in the power function in the fitness evaluation of a classifier, θ_{GA} specifies the GA threshold. ϵ_0 is the error threshold under which the accuracy of a classifier is set to one, and must be set according to the maximal payoff possible in an environment (usually to one percent of this number). θ_{del} is the experience threshold over which the fitness of a classifier may be considered in the deletion method, pX specifies the probability of applying crossover during a GA application, pM specifies the probability of mutating one attribute in the condition or the action of a classifier during a GA application, $P_{dontcare}$ is the probability of using (1.0, 0.0) in one attribute during covering, θ_{sub} specifies the required experience of a classifier to be able to subsume other classifiers and r_0 is the covering range used in the creation of hyperrectangular conditions $(l_i = f_i - r_0, u_i = f_i + r_0)$. For detailed explanation of these parameters, see [9].

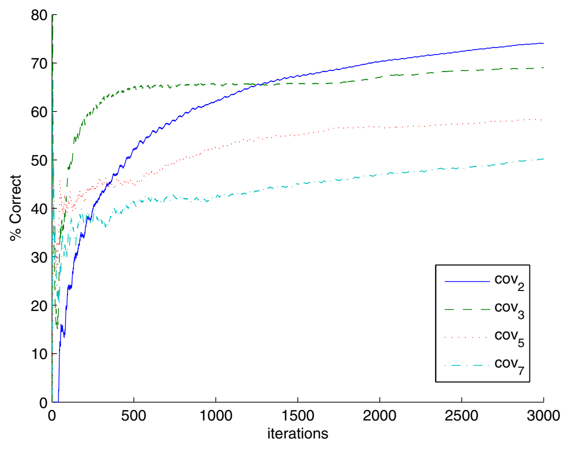

Fig. 3. System performance with various covering ranges. $cov_2 = 0.2, cov_3 = 0.3, cov_5 = 0.5, cov_7 = 0.7$

By default, the experiments used an equal ratio for exploration and exploitation. Each mode introduces one input at each phase but fitness update and GA only takes place in exploration mode. The experiments also used subsumption of classifiers. The maximum population limit N is set depending on the size of the input data of each experiment.

To test the learning capability, we study the number of speakers the system can learn under various encoding schemes. We present results from both closed-set and open-set experiments. All of our experiments are conducted at least ten times, each time using a different random seed and the results shown in graph are averaged.

A. Covering range

When no matching classifier exists in the present population for the current input, the system calls covering and introduces a new classifier that covers an area including the point specified by the current input. Covering range decides how much space the new classifier will cover. If the covering range is appropriate, it would cover enough area so that new inputs for the same speaker are included while excluding the inputs from other speakers. As shown in Fig. 2, the range of our voice feature is roughly -3.0 to 2.0. To see different behaviors of the system, we tried covering ranges from 0.1 to 0.7. Fig. 3 shows the performance depending on covering range. For this experiment, a covering range of 0.2 produced the best performance. Values between 0.2 and 0.3 have shown to be best so far, but it remains to be seen if this holds for any N arbitrary speakers. Automating the way to find the right covering range can be further investigated.

B. Dealing with complexity; Effects of Mutation/Crossover

A primary challenge in our use of XCS for speaker identification is the vast scope of the search space: cepstral data with 14 dimensions, classifier population sizes reaching more than half a million, and at least 6 digits of precision. As mentioned in section V-B, this space contains on the order of $(10^6)^{14}$ members.

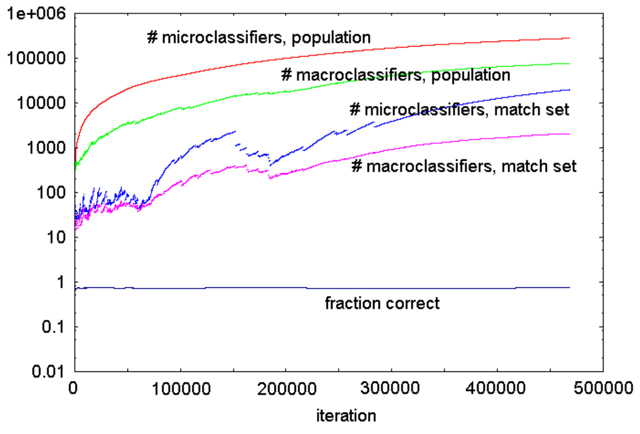

Fig. 4. Match set trendlines

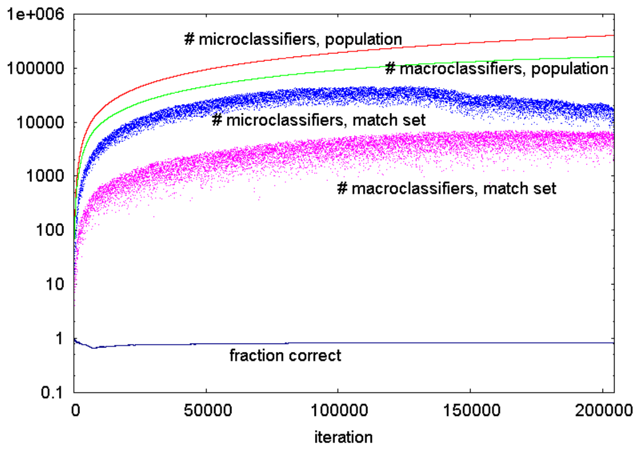

Fig. 5. Growth of population and matchset (45 speakers, 350 vectors/speaker)

Attempts to use earlier versions of our modified XCS brought much of the above complexity to our attention. A set of 80 input vectors produced over 50,000 macroclassifiers and was barely passing 400,000 iterations after a few months of Pentium 4 computation. This is in comparison to a 3,000-macroclassifier population and 500,000-iteration experiments in [7].

Given the desire to process many more than 80 input vectors at a time, and based on suggestions in [2], our experiments turned to disabling mutation and crossover. When the parameters pX and pM are set to zero, the system can avoid creation of many thousands of extra items in the classifier list, at the cost of possibly discovering more ideal combinations of speaker vector data and the resulting speaker choice. As a result of this tradeoff, the system gains significantly in computation speed, but loses only a few percent of correctness. The classifier list is also smaller and easier for a human to analyze for patterns.

One pattern of interest is that the system contains classifiers which are the "anti-answer" to the speaker identification question. That is to say, although these classifiers give the wrong answer from our perspective (not the correct speaker for the vector presented), they accurately predict that the system will give zero reward if the anti-answer is chosen. Precedent for such patterns was noted in [2]. This could possibly be put to use in a multi-step identification scheme.

While the no-mutation/crossover tradeoff was sufficient to allow timelier experiments on 900 input vectors (45 speakers, 20 vectors each), this was not the case for an even larger input set: 12,000 input vectors from 100 speakers (120 vectors each). Attempts to have XCS process these enormous datasets brought a return to the O(months) experiment rate, with 500,000 macroclassifier populations causing out-of-memory crashes and even fewer iterations completed.

In order to further reduce the search space, we modified the covering method involved in match set creation. Since the system is initially in a training mode, and the trainers know the correct input-action pair for each vector, the system is given this information during the covering step. This allows the system to only create new classifiers for a single possible action per input, rather than broadening the search space into input-action pairs for every other possible speaker as well as the expected speaker. Disabling the creation of 'default action' classifiers cuts another 50% of population size.

At this stage the system is back to timeliness for very large input set sizes. However, we now start to run into another problem - the input data itself. For a covering range of 0.2, the pared-down system creates a macroclassifier population about two-thirds the size of the input set. Interestingly, the number of classifiers associated with each speaker is roughly equal - it might be expected to decrease for later speakers as more of their vectors get absorbed into previously-covered classifiers (See section V-B).

Fig. 4 shows an example of the match set distribution over the course of an experiment. The distribution is composed of intermingled trendlines for each input vector, which follow the general pattern of changes in the overall population. Of interest are the trendline breaks, where the population is decreasing along with the match set size. The system performance does not decrease during these phases, suggesting that further beneficial generalization and reduction of the population is possible.

The tapering variance of the feature vectors, noted in IV-A, suggests that reducing the feature vector dimensionality could further speed up the training and testing phases of SID-LCS without reducing performance. Indeed, as displayed in Fig. 5, initial experiments in this vein show upwards of 10% improvement in correctness for large input sizes, brought about by the ability to reintroduce mutation and crossover without sacrificing timeliness. Disabling action mutation has also shown slight benefits here.

We made an attempt to gauge the data similarity of 125,000 of our input vectors. The system was set to behave as if all 125,000 vectors were for a single speaker. The resulting macroclassifier population was one-third of the input list size, with a fairly consistent classifier creation rate across the input iterations. We believe that the overall data "self-similarity" will be a deciding factor in XCS performance on the SID problem.

C. *Effects of Various Encoding Methods*

Butz [7] shows that different encoding methods can show much better performance according to their dependencies on dimension. Because our voice features do not show clear di-

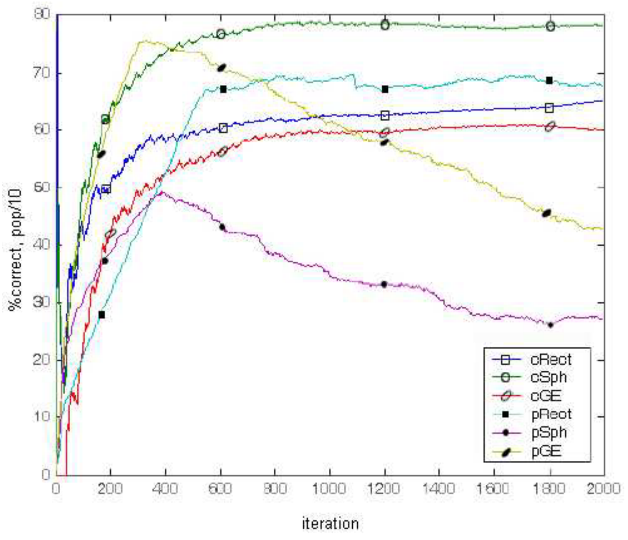

Fig. 6. Comparisons between various encoding methods (4 males, 20 feature vectors for each).

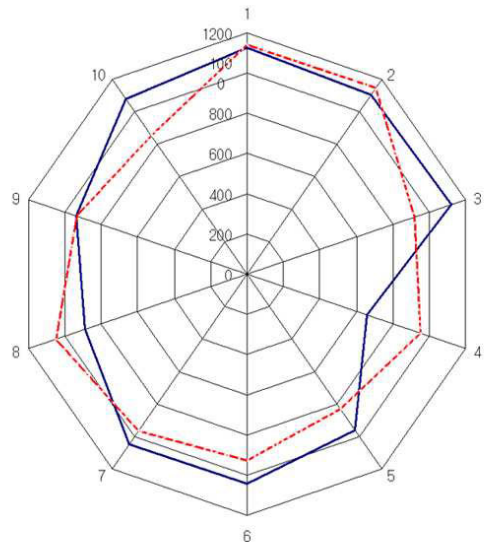

Fig. 7. Decision distribution. (10 males, 20 feature vectors for each). Hyperrectangular (solid) vs. Hyperspheroidal (dashed).

mensional dependencies, we tried several different encoding methods and observed the system performance. Three encoding methods are being tested with the SID-LCS system: hyperrectangular, hyperspheroidal and general hyperellipsoidal encoding methods. Fig. 6 shows the percent correct and the population size in tens for each encoding method over iterations. As shown in the graph labeled cSph, the SID-LCS with hyperspheroidal conditions can correctly identify 80% of 80 feature vectors from four speakers. Graph pSph shows that the hyperspheroidal encoding method needs a smaller number of rules than other encoding methods, converging to about 300 rules around iteration 2000. However, varying performance differences for larger numbers of speakers and vectors have left open the choice of encoding method.

To see if the different encoding methods have the same difficulties in classifying certain speakers, we compared the decision distribution between speakers under different encoding methods. Fig. 7 shows the decision distribution among 10 speakers. Each axis is for each speaker, with the number of times the system made a decision for the speaker. In the ideal case (100% correct), system decisions would be equally distributed as 1000 times for each speaker. The hyperrectangular encoding method, plotted by the solid line, has made less than 550 decisions for Speaker4 which let us assume that the classifiers for Speaker4 are not strong enough. However, for hyperspheres, plotted by the dashed line, decisions for Speaker4 are closer to ideal. Also, we can see that for Speaker1 and Speaker2, both of the encodings have strong classifiers. This shows that some speakers are easy to recognize regardless of the encoding methods, while some speakers are sensitive to encoding methods. This result leads us to some interesting ideas for our future research. For example, dynamically choosing the encoding method in the learning phase can be considered.

D. *Open-set Test*

For open-set classification, a set of 10 male speaker voice vectors are chosen as in-set speakers, randomly from the TIMIT data. The system is then trained with the ten in-set speakers and tested with 20 speakers which includes the 10 in-set speakers and 10 additional speakers as an out-set. As discussed in Section V-B, how many vectors would be enough for the training is still in question. In fact, this is a general concern of most machine learning systems. Computational learning theory [30] mathematically studies the relationships between the size of training set and the quality of solution. So far, we have trained the system with 20 vectors per speaker for open-set testing.

TABLE I
FALSE REJECT AND FALSE ACCEPTANCE FOR 200 OPEN-SET CASES.

	False Reject	False Acceptance
Set1	29 (14.5%)	48 (24.0%)
Set2	25 (14.5%)	65 (32.5%)
Set3	25 (12.5%)	59 (29.5%)
Set4	29 (14.5%)	53 (26.5%)

We have conducted experiments using four different sets of data. Table I shows the results. FA (False Acceptance) represents the number of out-set vectors that are incorrectly classified as in-set. FR (False Reject) is the number of in-set vectors that are mistakenly classified as out-set speakers. As shown, the False Reject (FR) rate is roughly half the False Acceptance (FA) rate. This is desirable in many applications since rejecting legitimate clients can cause frustration for valuable customers. All open-set experiments use hyperspheroidal encoding. The critical factor of this open-set classification is to present enough vectors that have sufficient information about the speaker in the training phase. So far, we only tried 20 vectors per speaker for more timely training. However, experiments with hundreds of vectors per speaker are now in progress.

E. Voting Method

The voting method, discussed in section V-E, significantly improves the system performance. We trained the system with 10 male speakers, 20 vectors each. In the testing phase, we initially introduced 10 vectors to the system and let it request more vectors up to 20 if needed. Using the voting method, the system was able to correctly identify 9 of 10 speakers using 20 vectors in 6 cases, and 10 vectors in 4 cases. Without the voting method, it only showed 73.6% correct with the same set of 10 male speakers. The number of vectors for adequate voting performance is still difficult to know. Notice that 20 vectors out of 250 feature vectors of 4 second voice sample are still relatively small number of vectors; we believe that considering more vectors to voting can improve the performance.

VII. Conclusions and Future Work

We have presented a new text-independent open-set speaker identification system using LCS. In doing so, we have investigated effective ways for applying XCS to the complex problem space of SID and observed its behavior in various situations.
Experiments are conducted for the closed-set and the open-set speaker identification problems. The results are comparable to recent statistical clustering methods [3]. The results show that LCS is learning effectively in a complex problem space when provided with new mechanism. We also confirmed that for the speaker identification problem, the performance of each encoding method, such as hyperrectangles, hyperspheroids and hyperellipsoids, varies over different speaker sets.
There are several future research directions for improving system performance. For one, encoding methods can be improved by investigating the relation between feature vectors and each encoding method. Deciding how many vectors are enough to train the system is one of the important questions to answer. For another, combining LCS with a case-injected system [31] can be considered. The system can build case-based memory for more efficient discriminative classification. We can also build a hybridized system with SID-LCS and numerical clustering, for both codebook specification and testing methods. Since SID-LCS provides a complete map that contains not only *'what to do'* but also *'what not to do'*, it can help numerical methods to make more accurate decisions.

Acknowledgments

Special thanks go to John Grieco, John Parker, Mark Huggins, and Darren Haddad of the Audio group at AFRL/IFEC, Rome, NY, USA.

References

[1] L.B. Brooker, D.E.Goldberg, and J.H.Holland, *Classifier systems and genetic algorithms*, Artificial intelligence, 40:235-282 (1989).

[2] S.W. Wilson, *Classifier fitness based on accuracy*, Evolutionary Computation, 3(2), 149-175 (1995).

[3] Pongtep Angkititrakul, John H.L. Hansen, and Sepideh Baghaii, *Cluster-dependent modeling and confidence measure processing for in-set/out-of-set speaker identification*, ICSLP (2004).

[4] J. C. Bezdek and P. F. Castelaz, *Prototype classification and feature selection with fuzzy sets*, IEEE Transactions on Systems, Man, and Cybernetics SMC-7 (1997), no. 2.

[5] Larry Bull, *Learning classifier system : A brief introduction*, Springer, 2004.

[6] Christopher Stone, Larry Bull, *For Real! XCS with Continuous-valued inputs*, Learning Classifier Systems Group Technical Report, UWELCSG02-007.

[7] Martin V. Butz, *Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system*, GECCO June 25-29 (2005), 1835–1842.

[8] Martin V. Butz and Stewart W. Wilson, *An Algorithmic Description of XCS*, Tech. Report 2000017, Illinois Genetic Algorithms Laboratory, 2000.

[9] Martin V. Butz. *XCSJava 1.0: An implementation of the XCS classifier system in Java*, Tech. Report 2000027, Illinois genetic Algorithms Laboratory, 2000.

[10] Arthur L. Corcoran and Sandip Sen, *Using real-valued genetic algorithms to evolve rule sets for classification*, IEEE Conference on Evolutionary Computation (Orlando, FL), 1994, pp. 120–124.

[11] T. M. Cover and P. E. Hart, *Nearest neighbor patter classification*, IEEE Transactions on Information Theory IT-13 (1967).

[12] Kenneth A. De Jong, *Genetic-algorithm-based learning*, Machine Learning, 3:611-638.

[13] Stephanie Forrest, *Implementing semantic network structures using the classifier system*, 1985, pp. 24–44.

[14] Richard Ricart, Jim Cupples, and Laurie Fenstermacher, *Speaker Recognition in Tactical Communications*, in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Adelaide, Australia 19-22 April 1994, p. I-329-I-332.

[15] John Grieco, *Advanced short segment language identification*, Tech. Report AFRL-IF-RS-TR-2005-344, Air Force Technical Report, 2005.

[16] J. H. Holland and J. S. Reitman, *Cognitive systems based on adaptive algorithms*, Pattern-Directed Inference Systems (D. A. Waterman and F. Hayes-Roth, eds.), Academic Press, New York, 1978.

[17] J.H. Holland, *Adaptation*, In R. Rosen & F.M. Snell (Eds) Progress In Theoretical Biology 4. Plenum.

[18] J. M. Keller, M. R. Gray, and J. A. Givens, *A fuzzy k-nearest neighbor algorithm*, IEEE Transactions on Systems, Man, and Cybernetics SMC-15 (1985), no. 4.

[19] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki, *The det curve in assessment of detection task performance*, EuroSpeech 1997 Proceedings 4 (1997), 1895–1898.

[20] A.D. McAulay and Jae C. Oh, *Improving learning of genetic rule-based classifier systems*, IEEE Transactions on Systems, Man, and Cybernetics 24 (1994), no. 1, 152–159.

[21] L. R. Rabiner and B. H. Juang, *Fundamentals of speech recognition.*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

[22] Xiaoyu Zhang Richard Mammone and Ravi P. Ramachandran, *Robust speaker recognition: A feature based approach*, IEEE Signal Processing Magazine (1996).

[23] A. E. Rosenberg, J. DeLong, C. H. Lee, B. H. Juang, and F. K. Soong, *The use of cohort normalized scores for speaker recognition*, Proc. ICSLP '92, 1992, pp. 599–602.

[24] S. F. Smith, *A learning system based on genetic adaptive algorithms*, Ph.D. thesis, University of Pittsburgh, Pittsburgh, PA, 1980.

[25] F. K. Song, A. E. Rosenberg, and B. H. Juang, *A vector quantization approach to speaker recognition*, AT & T Technical Journal 66 (1987), no. 2.

[26] DARPA TIMIT speech database, *http://www.mpi.nl/world/tg/corpora/timit/timit.html*.

[27] Stewart W. Wilson, *Get real! XCS with continuous-valued inputs*, Festschrift in Honor of John H. Holland (L. Booker, Stephanie Forrest, M. Mitchell, and Rick L. Riolo, eds.), Center for the Study of Complex Systems, 1999, pp. 111–121.

[28] Riolo, R. L., *CFS-C: A Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-Defined Environments*, Logic of Computers Group, Division of Computer Science and Engineering, 1988.

[29] Kovacs, T., *XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions*, Soft Computing in Engineering Design and Manufacturing, Roy, Chawdhry, and Pant, Eds., London, U.K.:Springer-Verlag, 1997, pp.59-68.

[30] *learning with irrelevant attributes*, Proceedings of the 35rd Annual Symposium on Foundations of Computer Science, Aditi Dhagat and Lisa Hellerstein:IEEE Computer Society Press, Los Alamitos, CA, pp.64-74, 1994.

[31] *Trap Avoidance in Strategic Computer Game Playing with Case Injected Genetic Algorithms*, Evolutionary-Computing Systems Lab, Department of Computer Science, University of Nevada, Reno - 89557.