J Sign Process Syst (2012) 68:151–170
DOI 10.1007/s11265-011-0594-x

A Low-rate Data Transfer Technique for Compressed Voice Channels

Vitaliy V. Sapozhnykov · Kurt S. Fienberg

Received: 16 September 2009 / Revised: 3 May 2011 / Accepted: 3 May 2011 / Published online: 31 May 2011

Abstract This study considers the problem of transmitting generic (non-speech) data through compressed voice channels such as those used in wireless communications networks. These networks employ voice codecs that are designed to efficiently encode and reproduce the relatively slow-changing signals of human speech, which leads to communication channels that are nonlinear and have long-term memory. A data modem is presented that utilizes finite alphabets of waveforms that are numerically optimized to be as separable as possible after passing through the voice codec. The optimization of the finite alphabet is performed using a pattern search algorithm. When used with the GSM Enhanced Full-Rate Voice Codec, this system demonstrated improved performance, in terms of error rates, compared to previously reported results. Simulation results for four other voice codecs are also presented.

Keywords Mobile communications · Data communications · Vocoders · Optimization methods

1 Introduction

The GSM [1] remains by far the most ubiquitous and internationally accepted wireless cellular network currently used by over 3 billion people in more than 212 countries. The GSM Association estimates that currently approximately 82% of the global mobile market uses the GSM standard: of which roughly 70% belongs to GSM 2G and 12% to UMTS [2]. Previous research [3–5] has shown that it is possible to communicate data through the GSM enhanced full rate voice codec at data rates sufficient for bursty, low-data rate applications. This would allow machine-to-machine applications over the widely used GSM mobile phone networks using the voice channel only. While dedicated data channels are now widely available, using the compressed voice channel for data transmission retains key advantages for some specific applications. Firstly, there is the wide coverage of legacy voice networks, especially in rural areas and developing countries where dedicated data networks have not yet become ubiquitous. Secondly, voice channels have quality of service (QoS) guarantees that do not apply to the common wireless data services, and in general voice calls are prioritized over data connections on networks where both exist, meaning that using the voice channel is more reliable when the network is congested. For example, the most common data service that coexists with GSM voice is the general packet radio service (GPRS) [6], and studies have shown that GPRS has queuing delay and outage probability that increase with increasing voice traffic on the network, leading to poor quality of service for GPRS in busy periods [7–9]. GPRS also has non-continuous coverage and interoperability issues [10, 11]. In addition, GPRS has a very high latency: a typical round-trip delay is 600–700 ms but may reach 1 s. Of course, both the reliability and capacity of data services such as GPRS can be increased through the upgrade of networks

V. V. Sapozhnykov (✉)
46 Brady Road, Bentleigh East, VIC 3165, Australia
e-mail: vvsapozhnykov@gmail.com

K. S. Fienberg
1/30 Swindon Road, Hughesdale, VIC 3166, Australia
e-mail: kfienberg@gmail.com

(for example to enhanced GPRS (EGPRS) [12, 13]), and coverage can be increased through wider rollouts, but these require significant investment [14], and have not yet reached many parts of the world, especially rural areas of developing countries.

Thus data transmission through existing wireless voice networks has the advantage of higher coverage and reliability. It certainly would not be seen as a replacement for dedicated data channels like GPRS, especially given relatively lower data rates, but as a useful option for certain applications where wider coverage and higher call priority are valued above high data rate, or for use in developing countries where no data service is present. If implemented by carriers it could also be used to balance their system load between data and voice traffic.

A data modem of the type described here would be useful in applications where highly reliable transmission of low rate data is required. Previous published research has looked at using data transmissions over the GSM voice channel for encrypted secure voice [4, 5, 15], since in the GSM network security is provided by the network operator, is not end-to-end for the user, and is generally not considered very secure. Another application presented in the literature is communicating Point-of-Sale (POS) transactions between the POS terminal and financial host [16]. Other possible applications for such technology include telemetry, automated meter reading, alarm systems, vehicle tracking/fleet management, security systems, and financial applications for wireless automatic teller machines (ATM). Taking the last as an example, the data-over-compressed-voice modem could be used for financial transactions as shown in Fig. 1, where the modem transmits transaction data between the ATM and bank financial hosts through the GSM mobile voice network. This would involve two data-over-voice modems of the type described in this study: one at the ATM itself and one between the GSM network and the bank host, in the location labeled “Exchange” in figure, which could connect to the Gateway MSC or media Gateway at the edge of the GSM cloud. Financial transactions are a suitable application for this type of modem because the volume of data is small (typically a few hundred bytes depending on transaction type and protocol) but reliability is important as a customer is generally standing at the ATM waiting on the transaction to complete. In this application the data over compressed voice modem would provide the following benefits when compared to existing wireline-based ATMs:

- Rapid roll-out: locations no longer require an expensive landline with long deployment times, but only being within GSM coverage.
- Low setup and maintenance cost.
- True mobility: units could be temporarily deployed for special events or locations could be changed to match customer use.

When compared to GPRS-based connected ATMs, the advantages of this technology would be:

- Higher reliability due to voice call priority and quality of service levels.
- Higher transaction throughput in highly congested areas: a transaction over the modem described here will happen sooner as voice has a priority over data.
- Wider coverage—ATM can be deployed where there is no cabling or wireless data channel.

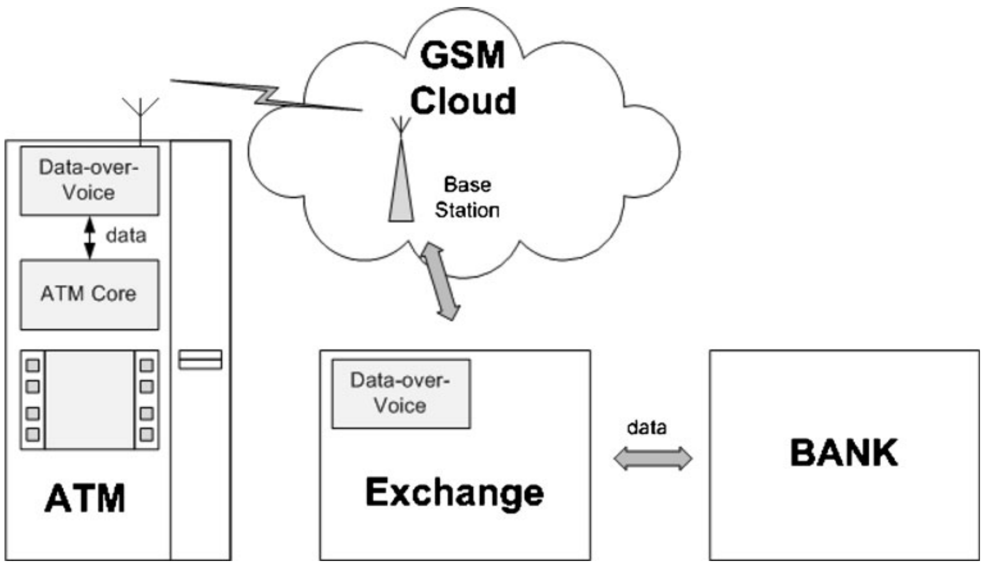


Figure 1 An example application for a data-over-voice modem is to transmit financial transaction data between automatic teller machines and bank financial hosts, through the GSM mobile network. This would involve two data-over-voice modems of the type described in this study: one at the ATM itself and one between the GSM network and the bank host, in the location labeled “Exchange” in figure, which could connect to the Gateway MSC or media Gateway at the edge of the GSM cloud.

The modem described here may also be used as a "backup" data channel for a case of the "main" data channel outage or unavailability. A great advantage of this "data over voice" type of system is that enabling a data service only requires GSM voice coverage. When a dedicated data channel is also available, a system to transfer data through the voice channel would allow options on voice/data traffic balance, and create a higher priority data stream for lower data rate but "mission-critical" applications.

In previous studies there has generally been two different approaches taken to modulate data into audio signals to pass through compressed voice channels. The first of these has been to map the input bit stream onto speech parameters of commonly used speech models, which are then used to synthesize an audio signal with speech-like characteristics. This was the approach taken in [4, 5, 15], where the speech models were Code Excited Linear Prediction (CELP) type, and the data was encoded into frame energy, pitch frequency, line spectral frequencies. In [17, 18] a similar method is proposed by mapping data onto the fundamental parameters including phases, frequencies and pitch frequencies related to formants in a speech-like waveform produced by an autoregressive speech model. Kotnik et al. [16] also used the parameters of an autoregressive speech model. These methods based on the use of speech models have the advantage of generating audio signals that are voice-like, and hence should trigger voice activity detection (VAD) systems in the voice network to identify the signal as voice (and therefore not cut off transmission as they would with "noise-like" signals used in traditional modulation schemes).

The second approach to transmitting data through speech codecs has been to encode the data into a finite alphabet of predefined symbols or waveforms that has been optimized for transmission through the voice codec [3, 19]. In [3] the symbol alphabets were optimized through an numerical search over all symbols of a fixed length that were within the bandwidth limits, using a genetic algorithm search procedure. On the other hand [19] generated the symbols by searching for the best performing symbols from amongst a database of waveforms derived from observed human speech. In these two studies the finite alphabet approach was shown to produce lower error rates and a simpler design, which facilitates implementation in a portable embedded device, and hence we adopt it for this study. The signal is not as "voice-like" however, and the VAD is generally not triggered by signals generated by mapping the input data onto finite alphabets of symbols, and if no further action is taken wireless voice networks can prevent transmission of the signal as it considered background noise only. But the GSM VAD system can be triggered by generating two different alphabets with different spectral shapes and alternating between them at regular intervals to create a dynamic spectrum [3].

While it has already been shown that data transmission through the GSM voice network is possible, the majority of previous studies by the authors [3] and others [4, 5, 19] have concentrated on the most commonly used voice coding scheme found in GSM the network—the GSM Enhanced Full-Rate (EFR) vocoder [20, 21]. While a pair of studies have examined data transmission through the full-rate GSM vocoder in [17] and [18]—even these choose a single codec to apply their approach. In reality however in the GSM network standard there are a range of four different voice codecs with different degrees of compression: the Half-Rate (HR) [22], Full Rate (FR) [23] and adaptive multi-rate (AMR) [24] vocoders, as well as the EFR vocoder, that can be chosen based on bandwidth requirements and desired speech quality [25]. For example a GSM network may use the EFR vocoder during periods of relatively light traffic to provide better speech quality but switch to HR vocoder when a cell is congested with high traffic, sacrificing voice quality for reduced bandwidth per call. In addition, other compressed voice network standards, e.g. VoIP networks, use different voice codecs, for example G729 [26]. Thus a general method for data-transmission over compressed voice channels, or even over just the GSM compressed voice channel, must be able to pass data through a range of different codecs. A plethora of different voice coding techniques have been developed over the last twenty years. Here, however, we will limit our research to the vocoders used in most ubiquitous applications: wireless cellular communications (such as GSM and UMTS) and Voice-over-Internet protocol (VoIP). To cover these applications, we will consider five voice codecs to communicate data through: EFR, FR, HR, AMR (used in GSM/UMTS), and G729 (used in VoIP) to prove that the proposed method can be extended to a range of speech codecs. It is expected, that in order to enable data transfer through any other compressed voice channel having a similar speech model, the same algorithm (reparameterized correspondingly) may be used.

So in this work we apply the general approach from [3] to several other voice codecs apart from just the GSM EFR vocoder, to assess its general utility, and to observe the effects of different compression rates and speech coding algorithms. Table 1 shows the list

Table 1 Vocoders used as communication channels for data over compressed voice simulations.

Vocoder	Bit rate, kb/s	CODEC type	Application
EFR	12.2	ACELP: Algebraic Code Excited Linear Prediction	GSM
FR	13.0	RPE-LTP: Regular Pulse Excitation—Long Term Prediction	GSM
HR	5.6	VSCELP: Vector Sum Excited Linear Prediction	GSM
AMR	Variable; 8 levels: 12.2 down to 4.75	ACELP: Algebraic Code Excited Linear Prediction	GSM/UMTS
G729	8	CS-ACELP: Conjugate Structure Algebraic Code Excited Linear Prediction	VoIP

of compressed speech codecs examined here and their respective bit rates as well as the speech model implemented by each vocoder.

Our general approach consists of using a finite alphabet (FA) of symbols to encode data into an audio stream, which is passed into a mobile GSM (or other voice network) unit exactly as if it were voice, where it is encoded by the speech codec. The signal is then passed over the air and through the voice network, and then decoded by the speech codec into a PCM stream once again. On reception, a maximum likelihood estimator (MLE) is used to decode the transmitted symbols of the signal. There are two requirements on the signal for this system to function: 1) it has to travel through the voice channel without setting off alarms such as a voice activity detection (VAD), which may shutoff transmission if the signal is considered noise and not speech, and 2) after passing through the vocoder's compression/decompression cycle, the symbols of the alphabet must remain sufficiently different that they can be distinguished by the MLE (at least to within some acceptable error tolerance). To resolve point 1, it was shown in [3] that regularly varying the spectral shape of the signal, achieved by swapping between two alphabets with different spectral characteristics (both within voice-band limits), was sufficient to deceive the GSM voice activity detection. Dealing with 2) is more difficult however, because most conventional methods of equalization [27] are not applicable to compressed voice channels. This is due to the fact that the vocoder is nonlinear in nature and has a long memory, and is practically impossible to represent by an analytic transfer function. In addition, the speech coding process involves multiple quantizations, which causes irretrievable loss of information about the original signal. Finally, the channel is acausal, as the codec operates on 20 ms blocks and therefore the channel effect at any point in time could be dependent on the signal up to 20 ms in its "future". The combination of these features of the compressed voice channel makes it very difficult for any conventional equalization scheme, such as used in modern communications systems [28, 29] to even approximately combat the signal distortions caused by a compressed speech channel (for a more in-depth description of a compressed voice channel see Section 2 in [3] and references therein). Faced with these difficulties for conventional methods, the requirement of robustness to the compressed voice channel was dealt with in [3] by using numerical optimization with a cooperative genetic algorithm to generate alphabet symbols that would be as separable as possible after a voice compression-decompression cycle.

In the current paper, however, the pattern search method (PS) [30–32] is proposed to solve the problem of generating a set of signals satisfying the given constraints. It will be shown that this produces symbol alphabets with lower error rates for transmission through the EFR voice codec than the genetic algorithm optimization presented in our previous work, as well as converging faster. This facilitates generating codebooks having better characteristics, which in turn, allows withstanding higher compression rates and therefore enables sending data through various different vocoders. The PS algorithm produces a finite alphabet of band-limited waveforms (symbols) as an off-line FA design routine. The generated FA is loaded into the modem and is used to communicate data through the compressed voice channel.

The paper is organized as follows. Section 2 outlines the basic approach taken and the general modem operation. Section 3 presents the alphabet generation process using the pattern search algorithm. Section 4 describes the results of the numerical simulations of the modem and compares them to previous studies. Section 5 discusses the results and considers directions for further investigation.

2 General Modem Description

2.1 Conceptual Approach

Due to non-linear nature of speech codecs, their acausality and long memory, it is practically impossible to describe the voice transcoding process in a closed form analytical expression. This means that the concept of conventional linear equalization (which assumes independence of the channel response from the input signal) similar to that of digital data communications is not applicable in this case. On the other hand, attempts to use non-linear equalizers [33] (e.g. neural network based equalizers) lead to necessity to accommodate potentially very large amount of training data which in turn implies the networks of practically intractable sizes ("dimensionality curse") i.e. networks too large to be implemented in portable embedded devices. Faced with these difficulties for conventional methods, the requirement of robustness to the compressed voice channel was dealt in [3] by using a "black box" approach.

According to this method, the channel vocoder impulse response or its inverse does not need to be identified and compensated for. Instead, data is communicated by means of a set of band limited waveforms, symbols, which maintain their separability (but not necessarily their original shape) after undergoing voice encoding/decoding process.

The rationale behind this approach is visualized in Fig. 2. Note that this figure demonstrates the concept in graphical form and is not meant to represent any particular symbol set generated. The multidimensional vector space C includes symbols a_i from a finite set A, $dim(A) = \{N, M\}$ as points (bold dots). Here N is the number of used symbols, and M is the symbol length. All possible vocoder outputs for each corresponding transmitted symbol a_i, $i = 1, ..., N$ is shown as an ellipsoid around each transmitted codeword. Note that the transmitted symbols a_i do not not necesarily sit at the cente of these ellipsoids of vocoder output, and that the size and shape of the probability distribution can vary between symbols. The area where the ellipsoids overlap corresponds to the error probability P_{err} between neighbouring symbols. It is more convenient to visualize the symbol mutual impact along curves l_i, each of which goes through the symbols of interest, and $i = 1, ..., L$, where L is the number of all possible symbol combinations. Each line l_i is a cross-section of possible output symbol overlap between adjacent symbols. N_i denote probabilities of symbol occurrence after passing the symbols through the voice codec. Errors will occur where these probability distributions overlap. Therefore, the greater the overlap is, the higher the error probability. It should be noticed that an error (misdetection) can potentially happen between any symbols a_i and a_j, $i, j = 1, ..., N; i \neq j$, from the symbol set A as the vocoder impact could theoretically map one symbol to any other, however closely related symbols are more probable to cause misdetection.

Therefore the goal of our communication system design is to find a symbol set A from the search vector space C such that probability distributions of the symbols a_i—members of this set—have minimal overlap after having undergone voice encoding/decoding process. This can be accomplished by numerical optimization. Once again, unlike conventional equalization, this "black box" method does not attempt to reconstruct a transmitted symbol by compensating for the channel impact. Instead, it only requires that the distances (in a chosen sense) between all the mem-

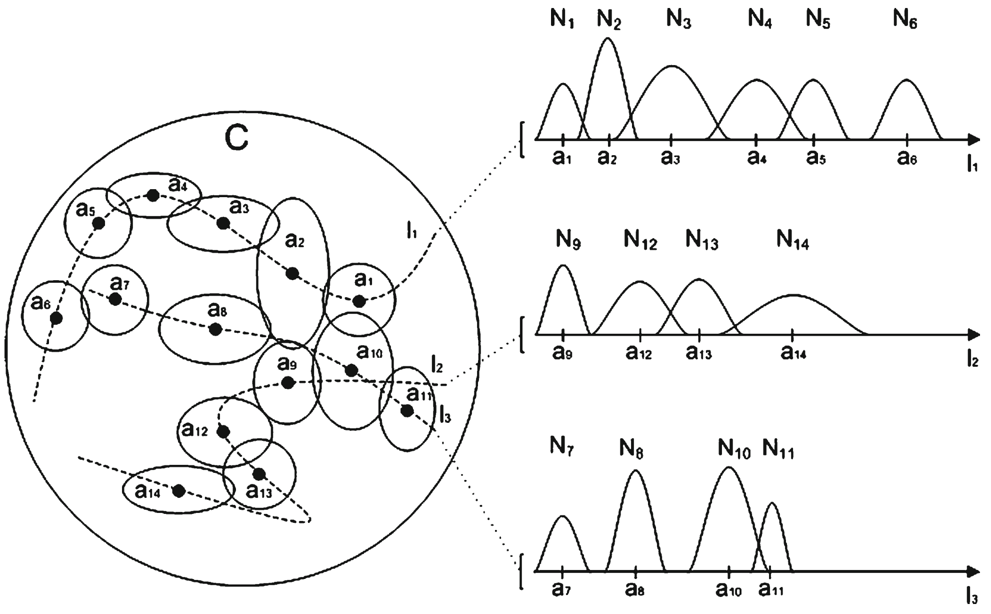


Figure 2 C represents an M-dimensional vector space which includes the finite alphabet A. Points a_i illustrate transmitted symbols of the finite alphabet A in the vector space C. The ellipsoids around each transmitted symbol represent the symbol variance due to voice transcoding process. On the *right* are the probability distributions N_i along the lines l_i.

bers a_i of A are affected as least as possible by the voice transcoding process. The knowledge of channel (the vocoder in this case) is therefore utilized in the "black box" sense: input/output relations. The vocoder is used to construct a set of waveforms (symbols) which attempt to preserve the distances between them while not necessarily preserving their original shape. Although time-consuming, the symbol-generating process is possible to accomplish offline because majority of voice codecs are standardized and available in software.

Once a Finite Alphabet of symbols is in generated, the general operation of the modem is straightforward. Firstly, an input data word is mapped onto a corresponding symbol by employing the FA as a look-up table. Then, the symbols are concatenated, framed and passed to the transmitting unit (TXU), which performs voice encoding, modulation and over-the-air transmission as per the corresponding standard. The signal received by the receiving unit (RXU) is demodulated and voice decoded in order to recover the received symbols. The estimate of the original data word is the index of the symbol from the FA which maximizes the likelihood of the received symbol. The TXU/RXU pair is an off-the-shelf component which implements one (or a few) of the existing communication protocols, such as GSM, UMTS, etc. Each protocol has its own modulation, equalization, error detection/correction technique to cope with multipath, channel noise, interference and other impairments. Therefore, it is assumed that for the modem under consideration transmission errors are stipulated by the voice coding/decoding process only. Field experiments have shown this to be a reasonable approximation. So, to enable data transfer through the compressed voice media, a simple mapper/demapper needs to be added to the existing off-the-shelf TXU/RXU.

2.2 Modem Operation

The general form of the modem proposed here to transfer data through a compressed voice channel is shown in Fig. 3. Data is mapped onto a set of waveforms—the symbols of our finite alphabet—which fit into the frequency range of the voice band and which are separable after a coding-decoding cycle through the compressed speech codec. The method for generating an alphabet (or codebook) of such symbols will be detailed in the next section, but for now let A be the alphabet composed of N such symbols, where each symbol consists of M samples.

The encoding process consists of grouping an incoming bit-stream of data into decimal words of size $N_b = log_2(N)$, then using the resulting decimal number to address the alphabet as a look up table and select the appropriate symbol to send. That is, a decimal scalar i is encoded into a vector symbol (waveform) $\mathbf{a}_i$, which is the i-th member of A.

The stream of symbols thus chosen are scaled (to fit a signed 13-bit PCM stream), concatenated and framed into packets, to form the transmit (TX) signal. This transmit signal is fed into the TXU of a compressed speech network, such as GSM voice or a VoIP system, exactly as if it were an audio speech signal. In the compressed speech network the signal is encoded using the speech codec, passed through the compressed voice network to its final destination, before being decoded by the speech codec and output as an audio stream once again. Note that the compressed voice network handles all the details of the over-the-air transmission, exactly

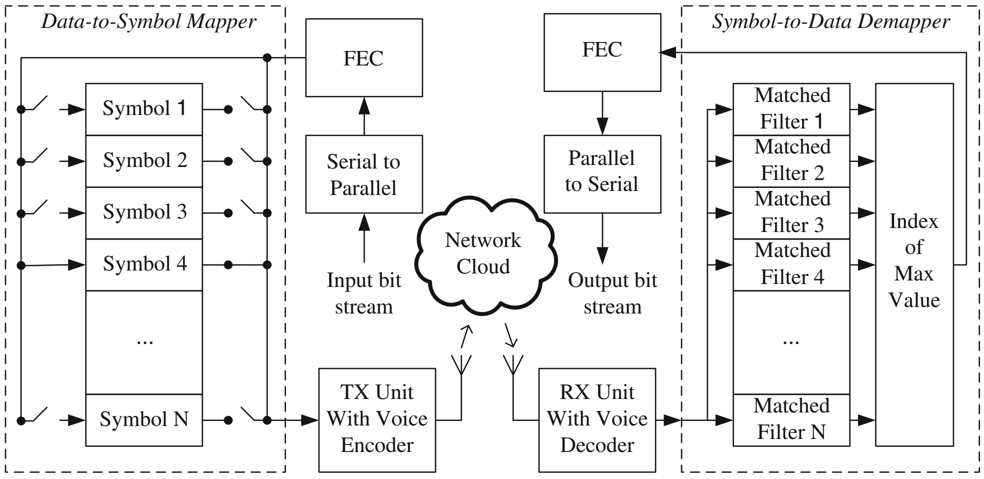


Figure 3 General approach to communicate data through compressed voice channels.

as it does when speech is transmitted (including channel equalization, error detection and error correction) so all of the distortion in our signal is taken to be the result of the speech coding, which assumes a voice signal with certain voice characteristics.

Upon reception, the audio output of the compressed voice network is then passed into the receiver (RX) side of our modem, which deframes the symbol packets and decodes the symbols. For each received symbol a maximum likelihood estimator is used to determine which, of the symbols in our codebook, is most probably the symbol that was transmitted. While we don't have an analytical form for the speech coder's transfer function, we can express it as the general operator $V(\cdot)$, given by:

$$\mathbf{y} = \mathbf{y}(\mathbf{a}_i, \psi) = V(\mathbf{a}_i, \psi), \tag{1}$$

where $\mathbf{y}$ is the received waveform that results from passing the original symbol $\mathbf{a}_i$ through the compressed voice channel, and $\psi = \Psi(\{\mathbf{a}_k\}_{k=1}^{i-1}, \psi_0)$ represents the internal state of the speech codec, which depends on the previous input symbols $\{\mathbf{a}_k\}_{k=1}^{i-1}$ and it's initial state upon reset ψ_0. In this notation the MLE then determines which data piece $\hat{i}$ was most probably transmitted by finding the maximum likelihood over the finite alphabet:

$$\hat{i} = \arg\max_i [P(\mathbf{y} \mid \mathbf{a}_i)], \tag{2}$$

where we have deliberately chosen to exclude the previous symbols of the history $\{\mathbf{a}_k\}_{k=1}^{i-1}$ from the likelihood function, taking into account dependence only on the current symbol, in order to reduce complexity and allow implementation in mobile embedded devices (see [3] for a more in depth discussion of this decision).

But Eq. 2 raises the question of what form is taken by the likelihood function $P(\mathbf{y} \mid \mathbf{a}_i)$. Simulation experiments through ACELP speech coding [34] indicated that $P(\mathbf{y} \mid \mathbf{a}_i)$ was ergodic, bell-shaped and of finite variance. It was not in general centered on $\mathbf{a}_i$, however, but for each symbol the central point of the distribution can be found by estimating the mean output value $\bar{\mathbf{y}}(\mathbf{a}_i)$, where

$$\bar{\mathbf{y}}(\mathbf{a}_i) = E\left[\mathbf{y}(\mathbf{a}_i, \psi)\right], \tag{3}$$

with $E[\cdot]$ is the expected value over all the voice codec states ψ. The value of $\bar{\mathbf{y}}(\mathbf{a}_i)$ is estimated by encoding random data into the symbols, performing the voice encoding/decoding process, and calculating the ensemble average on a symbol-by-symbol basis. This is done offline as part of the modem design process, using software simulations of the voice codecs, and not as part of the regular modem operation.

Then it is possible to approximate Eq. 1 for the received symbol as follows:

$$\mathbf{y} = \bar{\mathbf{y}}(\mathbf{a}_i) + \mathbf{n}(\mathbf{a}_i, \psi), \tag{4}$$

where $\mathbf{n}(\mathbf{a}_i, \psi)$ is the "effective noise" of the received signal, or more accurately the irregular part that depends on both the current vocoder state and the input symbol, not the input symbol alone. With these properties, we can subtract the mean value $\bar{\mathbf{y}}(\mathbf{a}_i)$ and the likelihood function $P(\mathbf{y} \mid \mathbf{a}_i)$ is approximated as a centered Gaussian process, with its variance dependent on $\mathbf{a}_i$. It can then be shown (see [3]) that the most probable symbol can be estimated by finding the maximum vector dot product between the received symbol $\mathbf{y}(\mathbf{a}_i, \psi)$ and the (normalized) precalculated mean symbols $\bar{\mathbf{y}}(\mathbf{a}_i)$, that is the decoding of the symbols is done by:

$$\hat{i} = \arg\max_i \left[\left\langle \mathbf{y}(\mathbf{a}_i, \psi), \bar{\mathbf{y}}(\mathbf{a}_i)\right\rangle\right], \tag{5}$$

where $\langle , \rangle$ represents vector dot product. This is a very simple decoding algorithm easily implementable on an embedded system as long as the size of the codebook is not too large (complexity of decoding scales as $N \times M$).

The index $\hat{i}$ is then the estimate of the transmitted data word i. These received decimal words can then be transformed back into a bitstream to recover (the best estimate of) the original bitstream. This completes the basic modem operation. Having described the general modem operation, we are now ready to proceed to the alphabet generation process.

3 Alphabet Generation Using Pattern Search

3.1 Introduction to the Pattern Search Algorithm

Pattern search is an optimization method often used to solve complex nonlinear problems. The PS algorithm performs optimization by sampling a search space in a "pattern" (a predefined set of points that is independent of the objective function), in order to determine a direction towards the minimum. This approach is known to have been successfully applied to solve optimization problems where analytical derivatives of the objective function are unavailable, unreliable or very expensive to compute. Furthermore, PS methods parallelize very well and therefore allow their implementation on parallel machines [35, 36]. This is particularly useful when the search space has a large number of dimensions.

A primary distinguishing feature of the pattern search methods is the way they sample the search space: the objective function is sampled over a predefined set (pattern) of points which lie on a rational lattice [37]. In [30] it was shown, that the global convergence of the PS method is guaranteed by imposing a structure on the form of the points in the pattern, as well as rules on both the outcome of the search and the subsequent updates. Thus, the pattern search method is chosen to generate a set of finite alphabets to communicate data through different voice codecs.

The most general form of the pattern search algorithm may be described as follows. Given a predefined pattern, a current point ("iterate"), and a step-size parameter, the search space is sampled by evaluating the objective function over points in a pattern around the current iterate. If one of the points in the pattern reduces the objective function to a value smaller than the current minimum, this point becomes the current iterate, the value of the objective function at this point becomes the current minimum, and the pattern is increased by the step-size. Otherwise, the current iterate remains the same and the scale of the pattern is reduced by the step-size parameter. The optimization process is stopped when the step size becomes smaller than some preset value.

3.2 Alphabet Generation as a Search Problem

In order to represent the alphabet generation as a search problem, it is necessary to condition the optimization process as follows. Firstly, the symbols that constitute the alphabet are generated in the frequency domain to fit into the designated voice bandwidth by design. Secondly, all the symbols in the FA need to be power normalized to achieve level insensitivity during the symbol-to-data demapping process. Thirdly, and most importantly, the generated symbols must be still identifiable (in the chosen sense), after having undergone the voice coding/decoding process. This robustness to voice compression may be expressed in terms of aggregate error rate (AER)

$$E_{AG}(A) = \sum_i E(\mathbf{a}_i) = \sum_i \frac{L_e(\mathbf{a}_i)}{L(\mathbf{a}_i)}, \qquad (6)$$

where $E_{AG}(A)$ is the aggregate error rate of the alphabet A, $E(\mathbf{a}_i)$ is the error rate of the ith symbol in A, $L_e(\mathbf{a}_i)$ is the number of erroneous detections out of $L(\mathbf{a}_i)$ times that the ith symbol of A underwent voice transcoding. The AER may be estimated by: encoding a large amount of random data into the symbols; transcoding them through a given voice codec; demapping symbols back to data; calculating the ratio of the number of erroneous symbol demappings to the total number of times passed through the vocoder for each symbol in the alphabet A; and summing these ratios.

It is convenient to define an objective function $f(A)$ for the alphabet optimization process in terms of AER, so that

$$f(A) = E_{AG}(A). \qquad (7)$$

Let band limits and unity power form the constraints on the search space, and the robustness to the voice transcoding (as measured by Eq. 7) be the objective function. In this way, the FA generation process may be formulated as a constrained nonlinear minimization problem.

With this in mind, it is now possible to list a set of components necessary for the FA generation process:

1. The Search space $\mathbb{S}$: a set of M-sample vectors constrained by
   – band limits
   $$\boldsymbol{\phi}_i \in [F_{\min},\ F_{\max}];\ i = 1, ..., N, \qquad (8)$$
   where $\boldsymbol{\phi}_i$ is the frequency spectrum of the ith symbol, $F_{\min}$ and $F_{\max}$ are the lowest and highest allowed frequencies respectively, and N is the number of symbols in the alphabet;
   – power unity
   $$\parallel \mathbf{a}_i \parallel = 1 \ \Rightarrow \ \parallel \mathbf{a}_i\, \mathbf{a}_j \parallel < 1;\ i, j = 1, ..., N;\ i \neq j. \qquad (9)$$
   where $\parallel . \parallel$ indicates the standard Euclidean norm.
2. The Objective function: a function $f(A)$ to be minimized, see Eq. 7.
3. The Search method: as was mentioned above, the pattern search method is chosen to perform the minimization of the objective function.

3.3 Alphabet Generation

In its most general form, the FA generation routine allows the following representation.

1. *Set an objective function and specify alphabet parameters*
2. *Generate initial alphabet*
3. *Apply pattern search method to optimize the initial alphabet*

Below is the detailed description of each step of the alphabet generation routine.

3.3.1 Set an Objective Function and Specify Alphabet Parameters

Setting the objective function requires a software implementation (or any other accurate model) of the corresponding voice codec. Fortunately, this is not a problem as the majority of the vocoders used nowadays are available in software (see [39] for example). For the alphabet generation purposes the objective function should be set up as described in Section 3.2 (see Eqs. 6 and 7). Alphabet specific parameters such as the number of symbols N in the FA, the number of samples per symbols M, and the number of active subcarriers K in a symbol are chosen empirically with the target data rate, error performance and symbol-to-data demapper complexity all being considerations. For example, to generate an alphabet which enables data transfer through the GSM EFR vocoder with the data rate of 4 kb/s the alphabet with the following set of parameters may be used:

- alphabet size $N \times M = 64 \times 12$;
- number of active subcarriers $K = 3$ (frequency spacing is $\Delta f = 666.67\ Hz$); total number of available subcarriers K_{total} is 4 ($1-4$, 0 being dc).

In [3] it was shown that in order to alleviate the VAD, two alphabets A^1 and A^2 need to be used per single TXU/RXU. These alphabets will be dynamically swapped at the TXU/RXU every 80 ms to allow dynamic change of the signal spectral envelope. Therefore, in this example, for the number of subcarriers $K = 3$ the subcarriers 1–3 are used to generate the alphabet A^1, whereas subcarriers 2–4 will be used for the A^2 generation.

With the objective function, the alphabet size and the number of active subcarriers specified it is now possible to outline the initial alphabet generation process.

3.3.2 Generate Initial Alphabet

The initial alphabet A_0 consists of N symbols $\mathbf{a}_i$, $i = 1, 2, ..., N$; each symbol $\mathbf{a}_i$ and its frequency domain equivalent $\boldsymbol{\phi}_i$ consists of M samples. The symbols are generated in the frequency domain in such a way that they are real in the time domain. This requires the real part of the complex symbol spectrum $\boldsymbol{\phi}_i$ to be even, and the imaginary part of $\boldsymbol{\phi}_i$ to be odd [40]. Let K denote the number of active subcarriers (active frequency bins), and k_N be the index of the subcarrier corresponding to the Nyquist frequency, so that $K < k_N < M$. The initial alphabet generation procedure is shown in Fig. 4.

This symbol generation process is similar to that in *orthogonal frequency division multiplex* (OFDM)

Figure 4 Initial alphabet generation.

Step 1. Initialize the index of the current symbol $i = 1$

Step 2. Randomly select a set of K complex numbers z_k, $k = 1, 2, ..., K$ from $\mathbb{R}^2 \supset [-1, 1] \otimes [-1, 1]$.

Step 3. Place z_k into the corresponding frequency bins (excluding dc) to produce the complex spectrum $\boldsymbol{\phi_i}$ of the symbol. The unused $k_N - K$ subcarriers are zero padded (ϕ_k is the kth subcarrier of the ith symbol)

$$\phi_k = \begin{cases} 0, & \text{if } k = 0\ (dc) \\ z_k, & \text{if } k = 1, ..., K \\ 0, & \text{if } k = K+1, ..., k_N \\ z^*_{M-k+1,} & \text{if } k = k_N + 1, ..., M \end{cases}$$

Step 4. Apply the inverse Discrete Fourier Transform to find the time domain symbol representation

$$\mathbf{g}_i = DFT^{-1}(\boldsymbol{\phi}_i).$$

Step 5. Normalize the symbol power to produce the final time domain symbol

$$\mathbf{a}_i = \frac{\mathbf{g}_i}{\|\mathbf{g}_i\|}.$$

Step 6. If $i = N$, then exit; otherwise let $i = i + 1$ and go to **Step 2**.

[41]. This way of generating alphabet guarantees that the symbols fit into the designated frequency band by design.

3.3.3 Apply the Pattern Search Method to Optimize the Initial Alphabet

In general, the alphabet optimization problem may be formulated as follows: *evolve the frequency content of each symbol in the initial FA to maximize the Euclidean distance between the symbols which have undergone transcoding over the chosen voice codec*. The frequency content implies a phase and a magnitude of each subcarrier comprising the symbol; the process of evolution is accomplished implicitly by the pattern search method in the frequency domain. Each symbol spectrum consisting of K active subcarriers is represented by a vector in $\mathbb{R}^{2K}$ (equivalent to $\mathbb{C}^K$). It can be transformed to the time domain through the process described in Fig. 4.

Let $\mathbf{x}$ denote the vector that results from concatenating the active Fourier bins from the frequency spectra of all N symbols in the current alphabet A. We can then represent the alphabet generation as a linearly constrained nonlinear optimization problem of the form

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{10}$$

subject to a constraint of

$$l \le \mathbf{x} \le u. \tag{11}$$

The objective function is $f:\mathbb{R}^n \to \mathbb{R}$, with the solution $\mathbf{x}\in\mathbb{R}^n$, dimensionality of the search space $n = N \times 2K$. In our case $f()$ is given by Eqs. 6 and 7, calculated by transforming the concatenated spectra $\mathbf{x}$ back into the time domain alphabet symbols A.

The bounds l and u are the vectors such that

$$l = [-1, -1, ..., -1],\;\; u = [+1, +1, ..., +1], \tag{12}$$

where $l, u \in \mathbb{R}^n$, so that the search is constrained to be within an n-dimensional hypercube. Let us denote by k the iteration index, $\mathbf{x}_k \in \mathbb{R}^n$ the current solution ("iterate" [37]), and $\Delta_k \ge 0$ the step-size parameter. Also the search algorithm requires the definition of a pattern of $P = 2 \times n$ search directions, which we denote by $D = [\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_P}]$. In order to ensure global convergence, D is required to be a positive spanning set for $\mathbb{R}^n$ in the sense of [37] and be composed of rational vectors. In [30] it was shown that the optimal choice of D is problem dependent. For the cases where no assumptions may be made about the search space structure, it often suffices to use the pattern $D = [I, -I]$, where I is a $P \times P$ identity matrix. Therefore $D \equiv [\mathbf{e_1}, \mathbf{e_2}, ..., \mathbf{e_n}, -\mathbf{e_1}, -\mathbf{e_2}, ..., -\mathbf{e_n}]$, where $\mathbf{e}_i$ represents the ith unit coordinate vector.

It should be noted, that the bound constraints are dealt with by searching over so called *feasible region* [38]: $\Omega = (\mathbf{x} \in \mathbb{R}^n : l \le \mathbf{x} \le u)$, that is by iterating over the set of directions that conform with the geometry of the boundary $[\mathbf{x}_k + \Delta_k \cdot \mathbf{d}_i] \cap \Omega$. For the bound constraints it is sufficient to check whether the points from the search set $\Theta_k = [\mathbf{x_k} + \Delta_k \cdot \mathbf{d_i},\; i = 1, ..., P]$ are within the bounds (lie between l and u), and iterate only over the feasible set $\Omega_k = \Theta_k \cap \Omega$ [38]. Recalling that the initial alphabet $\mathbf{x_0}$ is feasible by generation ($\Omega \supset \Theta_0$, $P_{\Omega_0} = P$, where P_{Ω_k} is the cardinality of $\Omega \cap \Theta_k$), it is clear that the kth search set Θ_k is determined by the structure of the search space.

The PS optimization strategy consists in identifying at each time instance k a current feasible search set $\Omega_k \subset \Omega$ and an associated solution vector $\mathbf{x}_+ \in [\mathbf{x}_k + \Delta_k \cdot \mathbf{d}_i,\;\; i = 1, ..., P_{\Omega_k}]$ which minimizes the objective function so that $f(\mathbf{x}_+) = \min(\mathbf{x}_k + \Delta_k \cdot \mathbf{d}_i),\;\; i = 1, ..., P_{\Omega_k}$. If such a point is identified, the step-size is increased to be $\Delta_{k+1} = 2\Delta_k$, and the vector which minimizes the objective function at the kth time instance becomes the next iterate so that $\mathbf{x}_{k+1} = \mathbf{x}_+$. Otherwise, if none of the points in the pattern $[\mathbf{x}_k + \Delta_k \cdot \mathbf{d}_i,\;\; i = 1, ..., P_{\Omega_k}]$ reduces the objective function, then for the next iteration the current iterate remains unchanged $\mathbf{x}_{k+1} = \mathbf{x}_k$, and the step-size is halved, so that $\Delta_{k+1} = \frac{1}{2}\Delta_k$. This process is repeated until the search terminates when the step size Δ_k becomes smaller than some preset tolerance. The PS algorithm is summarized in Fig. 5. In order to be loaded into the real hardware (for example an embedded system), the generated alphabet may be scaled up and converted to the fixed-point n-bit representation.

For high dimensioned search spaces such as used here (for a FA with 64 symbols and 10 active subcarriers the search space is 1,280-dimensional) the PS algorithm requires the search over a very large number of points per iteration, leading to high computational load and long alphabet generation time. However, because each location polled is independent the algorithm lends itself to parallel implementation, with the objective function at each search point being calculated by a separate processing unit [36]. This allows much faster search if parallel computing is available. For this study, the pattern search algorithm was implemented on the GPU based parallel processing unit nVidia Tesla C870 [42].

Figure 5 Pattern search algorithm for alphabet optimization.

```
Initialization:
• Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be an objective function (10)
• Let $l, u$ be a set of bound constraints as defined by (12)
• Let $D = [\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_P}]$ be a set of search directions
• Let $\Delta_0$ be an initial step-size parameter
• Let $tol$ be a stopping tolerance
• Let $\mathbf{x_0} \in \Omega$ be an initial solution alphabet generated as shown in Fig. 4
and calculate $f(\mathbf{x_0})$
• Let iteration index $k = 0$

Algorithm:
Step 1. Check feasibility: determine $\mathbf{d_i} \in D,\ i = 1, ..., P$ such that
        $(\mathbf{x_k} + \Delta_k \cdot \mathbf{d_i}) \in \Omega$

Step 2. Compute $(\mathbf{x_k} + \Delta_k \cdot \mathbf{d_i}),\ i = 1, ..., P_{\Omega_k}$ over the feasibility set $\Omega_k$ and
evaluate $f(\mathbf{x_k} + \Delta_k \cdot \mathbf{d_i}),\ i = 1, ..., P_{\Omega_k}$

Step 3. Determine $\mathbf{x}_+$, such that $f(\mathbf{x}_+) = \min(\mathbf{x_k} + \Delta_k \cdot \mathbf{d_i}),\ i = 1, ..., P_{\Omega_k}$

Step 4. If $f(\mathbf{x}_+) < f(\mathbf{x_k})$, then $\mathbf{x}_{\{k+1\}} = \mathbf{x}_+$, $\Delta_{k+1} = 2\Delta_k$; else $\Delta_{k+1} = \frac{1}{2}\Delta_k$

Step 5. If $\Delta_k > tol$, then $k = k + 1$, go to Step 1; else, exit
```

4 Results of Simulations

Performance of the proposed FA generation technique have been assessed by 1) generating a set of alphabets which provide various data rates, 2) encoding a large amount of random data into these alphabets, 3) performing voice transcoding of the symbols through the chosen set of vocoders, and 4) calculating symbol error rate (SER) and analyzing results. Design parameters for each alphabet are summarized in the Table 2. The alphabets are labeled in this table by their dimensions, i.e. the alphabet with N symbols of length M samples is labelled as $N \times M$, and we will continue to refer to them by this label. All the alphabets have been generated following the procedures described in Section 3.

Common parameters for all alphabets were: the sampling frequency $f_s = 8{,}000$ Hz and the designated bandwidth $B = [300,\ 3{,}400]$ Hz. The total number of active subcarriers K_{total} was chosen in such a way that the symbol spectrum would not exceed the given voice band B. The data rate R as shown in Table 2 is the determined from the alphabet parameters by

$$R = \frac{N_b}{M} f_s \text{ b/s.} \tag{13}$$

Figure 6 shows the convergence of the 64 × 12 FA. It may be seen that it takes approximately 8,000 iterations for the algorithm to converge to the error floor of 2.5%. The PS algorithm exposes exponential convergence with the most significant reduction of the cost function occurring within the first 50 iterations. On the other hand, as was shown in [3], a similar alphabet is generated by the cooperative GA in 25,000 iterations with the error floor of 3%. Therefore the PS algorithm described in this paper converges 3 times faster with a 16% performance improvement compared to the cooperative GA developed in [3]. It was noted that the convergence rate was approximately the same for all the alphabet sizes, although the error floors varied significantly.

The error performance for all the generated alphabets is listed in Table 3. Consider first the error rates when the number of symbols in the alphabet is fixed at $N = 64$, which are represented in Fig. 7. It may be seen that two highest rate alphabets—64 × 12 (4 kb/s) and 64 × 16 (3 kb/s)—can enable data communications through the EFR and AMR (level 12.2) vocoders, however are unusable over the vocoders with higher compression rates due to the high symbol error rates observed. When the data rate is reduced to 2kb/s by increasing the symbol length to 24 samples, the 64 × 24

Table 2 Alphabet design parameters.

$N \times M$	Rate, kb/s	Δf, Hz	K_{total}	Active Fourier bins
16×8	4.00	1000	3	1–3
16×11	2.91	727.27	4	1–4
16×16	2.00	500	6	1–6
16×24	1.33	333.33	9	1–9
16×32	1.00	250	12	2–13
16×42	0.76	190.48	16	2–17
32×10	4.00	800	4	1–4
32×13	3.08	615.38	5	1–5
32×20	2.00	400	8	1–8
32×30	1.33	266.67	11	2–12
32×40	1.00	200	15	2–16
32×53	0.75	150.94	20	2–21
64×12	4.00	666.67	4	1–4
64×16	3.00	500	6	1–6
64×24	2.00	333.33	9	1–9
64×36	1.33	222.22	13	2–14
64×48	1.00	166.67	18	2–19
64×64	0.75	125	24	3–26
128×14	4.00	571.43	5	1–5
128×19	2.95	421.05	7	1–7
128×28	2.00	285.71	10	2–11
128×42	1.33	190.48	16	2–17
128×56	1.00	142.86	20	3–22
128×74	0.76	108.11	28	3–30
256×16	4.00	500	6	1–6
256×21	3.05	380.95	8	1–8
256×32	2.00	250	12	2–13
256×48	1.33	166.67	18	2–19
256×64	1.00	125	24	3–26
256×85	0.75	94.12	32	4–35

N is number of symbols in the alphabet, M is the symbol length in samples, Rate is the throughput in kb/s, Δf is spacing of the bins in the Fourier domain for the given symbol length, K_{total} is the total number of available subcarriers per symbol, and Active Fourier gives which of those Fourier bins are used (with bin numbering starting from bin zero being the DC offset).

alphabet has error rates that are reasonably low for the FR codec as well as EFR and AMR 12.2, but again unusable on lower rate codecs. Similarly, reducing the throuput of the modem further the 64×36 (1.33 kb/s) can communicate data through the EFR vocoder, G729, AMR (levels 12.2 and 7.4); the 64×48 (1 kb/s) and 64×64 (0.75 kb/s) have reasonable error rates over all voice codecs specified here (the former has rather large SER over HR vocoder and AMR level 6.7).

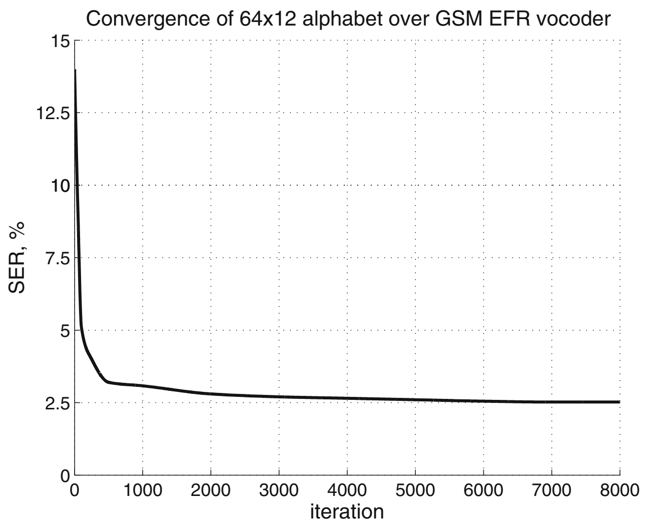


Figure 6 Convergence of 64×12 alphabet over GSM EFR voice codec.

For alphabets with significantly smaller number of symbols, Fig. 8 shows that when N is fixed at 16 symbols per alphabet the pattern of performance is the same, with longer symbols and lower throughput allowing lower symbol error rates. Once again this is most noticeable for the vocoders with higher levels of compression such as HR or AMR 6.7. However, although the general pattern is the same, the $N = 16$ alphabets show slightly higher symbol error rates across the board when compared to the $N = 64$ alphabets with the same data rate (not the same symbol length). So significantly decreasing the number of symbols per alphabet causes an increase in error rates of approximately 1.2 times. In a similar way we can consider the effect of significantly increasing the number of symbols per alphabet, which is shown in Fig. 9 where there are $N = 256$ symbols per alphabet. Once again the

Table 3 Performance of the alphabets over the different voice codecs and compression levels.

$N \times M$	Rate,kb/s	EFR	AMR 12.2	FR	G729	AMR 7.4	AMR 6.7	HR
16 × 8	4.00	3.2545	3.1485	19.656	51.149	54.666	85.763	89.638
16 × 11	2.91	0.4705	0.4763	9.732	18.063	21.954	54.653	69.154
16 × 16	2.00	0.0005	0.0009	1.917	9.851	10.360	42.430	52.224
16 × 24	1.33	0.0004	0.0007	0.447	1.174	1.438	20.502	23.649
16 × 32	1.00	0.0002	0.0005	0.158	0.281	0.293	5.803	9.790
16 × 42	0.76	0.0000	0.0000	0.052	0.051	0.095	1.560	4.173
32 × 10	4.00	2.8194	3.0970	19.178	50.302	58.235	86.146	83.954
32 × 13	3.08	0.3553	0.3806	8.803	15.970	20.836	49.493	59.366
32 × 20	2.00	0.0004	0.0008	1.867	9.755	9.303	37.786	48.970
32 × 30	1.33	0.0003	0.0006	0.383	1.119	1.370	19.457	19.462
32 × 40	1.00	0.0002	0.0004	0.145	0.266	0.253	5.609	8.413
32 × 53	0.75	0.0000	0.0000	0.049	0.052	0.082	1.294	3.873
64 × 12	4.00	2.5097	2.5495	15.365	42.127	45.006	72.504	72.002
64 × 16	3.00	0.3466	0.3670	7.604	13.942	17.503	42.677	53.331
64 × 24	2.00	0.0004	0.0007	1.516	8.212	8.024	33.030	40.827
64 × 36	1.33	0.0003	0.0006	0.342	0.933	1.130	15.767	19.028
64 × 48	1.00	0.0002	0.0003	0.125	0.230	0.217	4.423	6.910
64 × 64	0.75	0.0000	0.0000	0.044	0.043	0.075	1.145	3.303
128 × 14	4.00	2.3480	2.4278	14.459	37.680	41.353	65.615	66.689
128 × 19	2.95	0.3253	0.3424	7.327	13.047	16.185	39.752	50.067
128 × 28	2.00	0.0004	0.0007	1.354	7.729	7.598	30.644	37.269
128 × 42	1.33	0.0003	0.0005	0.325	0.870	1.050	14.852	18.105
128 × 56	1.00	0.0001	0.0003	0.115	0.215	0.211	4.140	6.324
128 × 74	0.76	0.0000	0.0000	0.042	0.039	0.070	1.071	3.063
256 × 16	4.00	2.1020	2.1455	13.358	36.970	38.757	60.242	64.792
256 × 21	3.05	0.2926	0.3033	6.483	11.788	14.716	36.502	43.456
256 × 32	2.00	0.0003	0.0006	1.276	7.018	6.698	24.650	34.260
256 × 48	1.33	0.0003	0.0005	0.286	0.794	0.953	13.156	16.160
256 × 64	1.00	0.0001	0.0003	0.105	0.195	0.186	3.756	5.814
256 × 85	0.75	0.0000	0.0000	0.038	0.037	0.064	0.980	2.862

Values under the codec headings are raw (uncoded) symbol error rates as percentages. Values of 0 indicate an error rate less than 0.0001% (10^{-6}).

overall pattern and relationships are maintained, with longer symbols (i.e. lower throughput) causing lower error rates, and lower bit rate codecs causing increased symbol error rates: all as one might expect. When we compare the symbol error rates for $N = 256$ alphabets with those of the $N = 64$ alphabets with the same data rate, we can see that the former are roughly 15% lower i.e. error rates are approximately 0.85 times those of the $N = 64$ alphabets with the same data rate. So in general we see that the more symbols per alphabet, the lower the symbol error rate, assuming the symbol length is varied to maintain the same data rate and that the same vocoder is used in both cases. Finally, this can be seen explicitly in Fig. 10 where we show the symbol error rates across various codecs as the number of symbols N is varied from 16 to 256. Note that in Fig. 10 only the alphabets with data rate of 2 kb/s are shown to allow a fair comparison. Again it can be seen that there is a decrease in symbol error rate with increasing N for all vocoder types. But this performance improvement comes at a cost: increasing the number of symbols per alphabet N will necessarily increase the computational complexity of the modem, specifically the demodulator, since the number of operations in the decoding of the symbols described by Eq. 5 is proportional to N. So to gain the 15% improvement in error rate that comes from increasing N from 64 to 256, the computational load of the final modem would have to be increased by a factor of four (this is only considering the decoding step as it is by far the most computationally complex).

Note that Table 3, as well as Figs. 7–10, show uncoded error rates. But in some cases where the symbol error rate is low but not trivial, the system can benefit from forward error correction (FEC) (for high error rates it is more efficient to simply reduce data rate than to add FEC due to the large overhead required to correct large numbers of errors). Reed–Solomon codes were proposed as an appropriate method for this in [3], due to fact that the Galois field order could be chosen to be the same as the symbol size in bits, meaning symbol errors would not spread into more

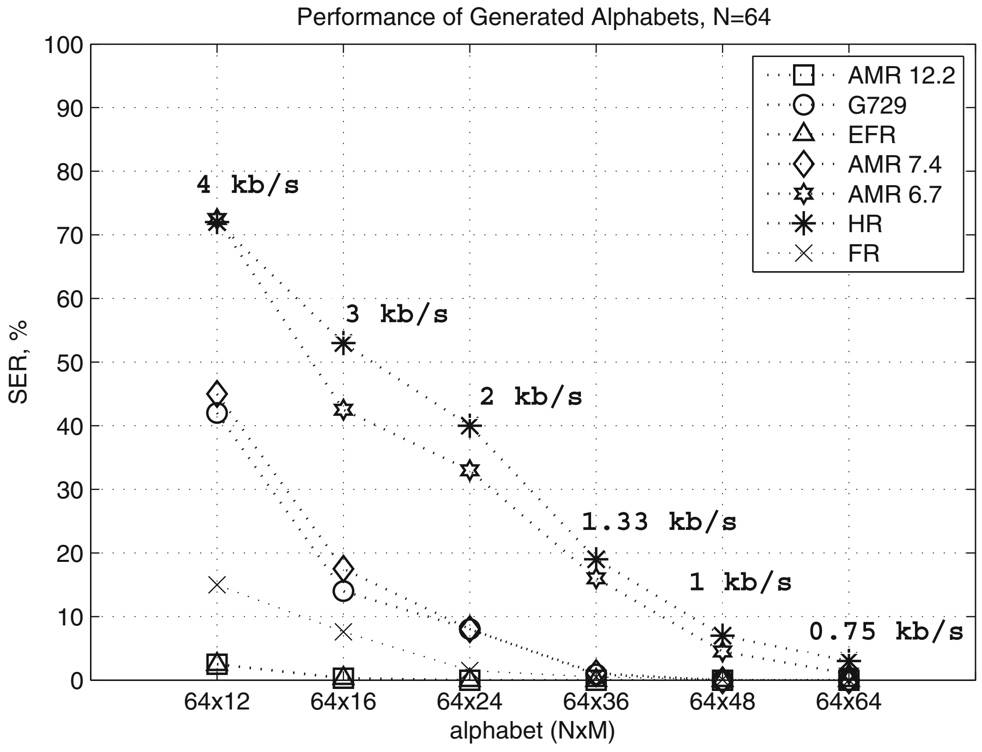


Figure 7 Performance of generated alphabets over various voice codecs when number of symbols is $N = 64$.

than one Reed–Solomon word, and hence improving FEC performance. The addition to the modem of a Reed–Solomon FEC scheme over a 6 bit Galois field with approximately 30% overhead was seen to be able to reduce a system with a 5% SER to an error rate of approx. 10^{-6}.

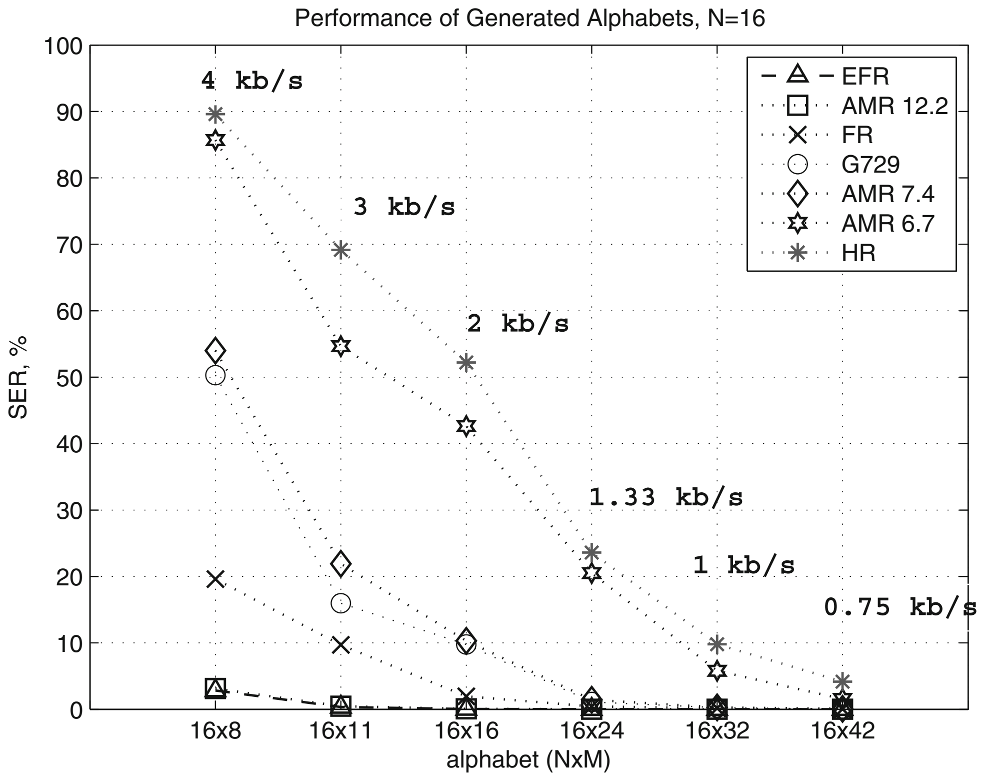


Figure 8 Performance of generated alphabets over various voice codecs when number of symbols is $N = 16$.

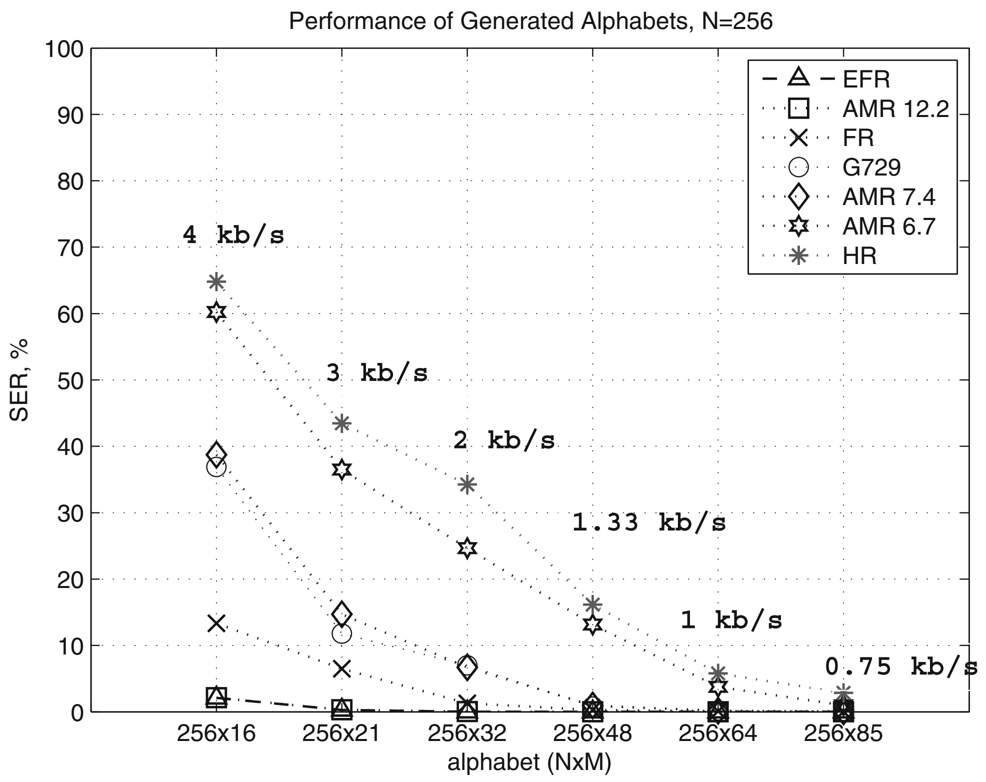


Figure 9 Performance of generated alphabets over various voice codecs when number of symbols is $N = 256$.

Figure 11 shows the maximum system data rate of the proposed modem as a function of the vocoder rate. This was found by taking highest data rate alphabet that gave a symbol error rate of less than 5%, chosen because this error rate could be corrected by the Reed–Solomon forward error correction scheme suggested

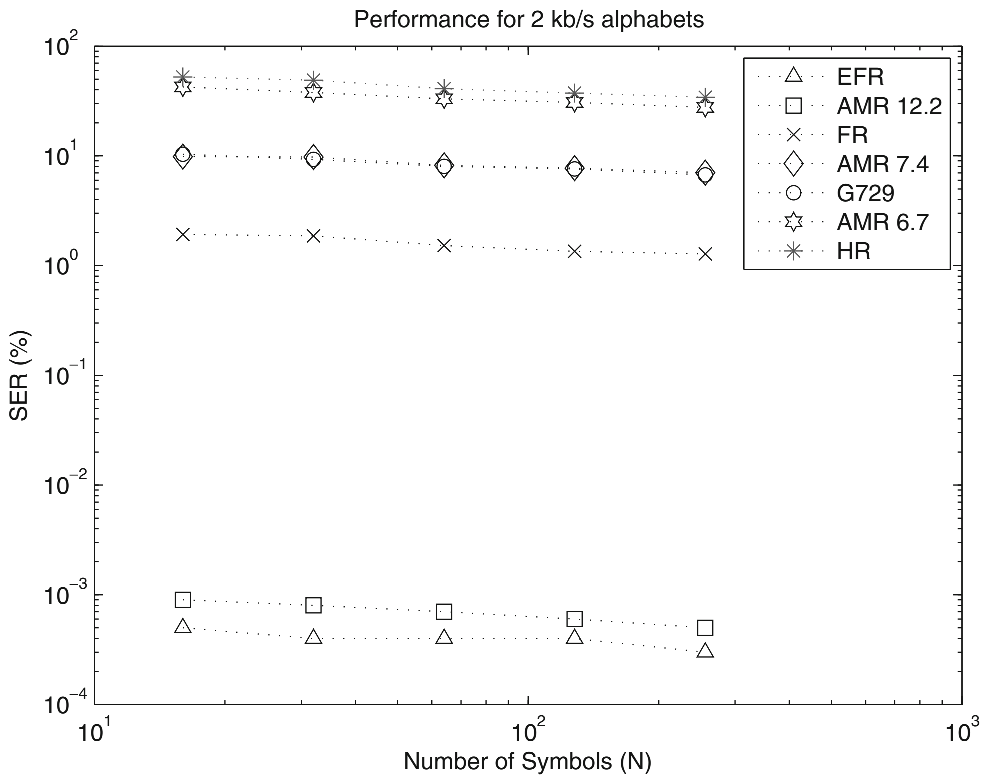


Figure 10 Performance of generated alphabets over various voice codecs for different number of symbols per alphabet N. All alphabets shown have data rate of 2 kb/s.

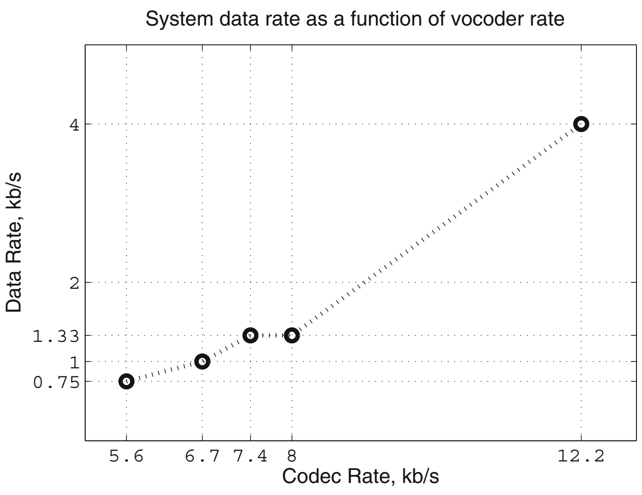


Figure 11 System data rate as a function of vocoder rate.

in [3]. As would be expected, the degree of compression applied to the speech signal is seen to significantly affect the data rate that was possible to communicate over the compressed voice channel, with EFR and the top level of AMR codecs, which compress speech down to 12.2 kb/s, allowing a data transmission rate of 4 kb/s with reasonable error levels, while the HR and low level AMR, which compress speech down to 5.6 and 6.7 kb/s respectively, allowing only 0.75 kb/s to pass through with a low symbol error rate. The only exception to this the FR codec, which has the highest bit rate at 13 kb/s but allows a data rate of only 2 kb/s, less than both EFR and the top level of the AMR vocoder. Note that FR was the first GSM vocoder, using RPE-LTP (see Table 1 and [23], which was replaced by the ACELP based EFR due to relatively poor (subjective) speech reproduction, so the differences seen here may be due to the inferior performance and audio reproduction of the FR codec compared to the ACELP based EFR and AMR 12.2 vocoders. In conclusion, apart from the FR codec the possible data rate with this method is a strong function of the degree of compression. As was mentioned earlier, this technique is not envisaged for multimedia applications which require high data rate. Instead, it is proposed to deal with low-rate bursty data transmissions where the reliability of data delivery is valued over data rate.

In [3] it was observed that the alphabet error performance is determined by the codebook diversity coefficient (CDC)

$$\eta = \frac{N_b}{M} = \frac{log_2(N)}{M}, \tag{14}$$

where N_b is the number of bits mapped onto a single symbol, M is the number of samples per symbol, N is the number of symbols in the FA. The data rate R of the system is then the product of the CDC and the sampling frequency, i.e. $R = \eta \cdot f_s$. As the data rate (and therefore, the CDC) increases, the symbol error rate gets worse, and vice versa, with the data rate (and the CDC) decreased, the symbol error rate improves. This trend was not altered by the new optimization method or considering different speech codecs. This is as expected, because the longer the symbols (and therefore the more active subcarriers comprise the symbols) for a given FA size, the higher the dimensionality of the signal subspace, so the larger the Euclidean distance that may be achieved between the symbols. However, as the symbol length is increased (for a fixed alphabet size N), the data rate decreases since the same number of bits is encoded into the longer symbols.

Overall simulation results may be summarized as follows.

1. It is possible to communicate data through all five vocoders tested, and all 7 compression levels. This is presumably due to the commonality of the speech models used to design the voice codecs.
2. In general, the higher the compression rate is, the lower the data rate has to be to provide reasonable error rates. Indeed, it was found that the higher the compression rate, the higher the level of the "equivalent noise" (see Eq. 4). Therefore, the length of the symbol and the number of the active subcarriers it contains have to be increased in response, as the dimensionality of the signal subspace determines the potentially achievable Euclidean distance between the symbols in the FA of a given number of symbols.
3. For the given vocoder the symbol error rate grows with increasing data rate (follows from above).
4. For a fixed data rate and vocoder, the symbol error rate decreases with increasing number of symbols per alphabet N.
5. As in some instances a non-trivial symbol error is exposed a forward error correction (FEC) should be applied. Reed–Solomon codes are suggested.

4.1 Comparison with Previous Results

As outlined in Section 1, there are two broad approaches that have been described in the literature for transmitting data over the compressed voice channel: the finite alphabet method that we have described here and the method of using a mathematical model of speech and mapping the data onto the speech parameters. Firstly we compare our results with other studies using the same general concept of finite alphabets. In our previous article [3] the same method of optimized finite alphabet was employed, except that the optimization was performed using a cooperative-competitive genetic algorithm, and the only vocoder considered was the EFR codec. As noted above, when generated through the pattern search algorithm the 64×12 alphabet (data rate of 4 kb/s) had a 16% symbol error rate reduction for the EFR codec relative to the alphabets generated in [3]. Alphabets of other dimensions saw a similar improvement in error rate reduction of 10-20% for the EFR vocoder. The only exception to this was for the alphabets with 2 kb/s of $N = 16$ and $N = 32$, where the improvement in error rate over that in [3] is several orders of magnitude. In addition the optimization algorithm itself converged a 3 times faster than the genetic algorithm previously employed. The computational complexity of the modem operation once the alphabet was generated is unchanged from that reported in [3] for a given alphabet size, since the modem implementation was identical except for alphabet generation (which can be performed off-line). The computational complexity of the demodulation, which is by far the most demanding operation, is approximately $2.2 \times 10^4 \dot{N}$ instructions per second, which is easily feasible on a modern embedded platform even for the maximum alphabet size considered here ($N = 256$). Thus the method outlined here produces lower symbol error rates and shorter alphabet generation times than that in [3] while maintaining the same computational complexity of the final modem operation. In addition we have also extended it to other vocoder types.

The finite alphabet approach was also taken in [19] to transmit data over the EFR vocoder. Their approach restricted the alphabet optimization to a search over a database of voice-like symbols derived from human speech. While reducing the search space in this way improves the speed of the search, it also seems to reduce performance, as the 2 kb/s alphabet generated in [19] had a raw uncoded symbol error rate of 1.5×10^{-5}, as opposed to $2 - 5 \times 10^{-6}$ for the various 2 kb/s alphabets generated here via pattern search over the Fourier domain of band limited signals. Note that the use of these voice-like symbols was not sufficient to trigger the voice activity detection, and [19] uses the same method of varying the spectral envelope with two alphabets to trigger VAD that we used in [3]. So no supplemental benefit was gained from the use of voice-like symbols (apart from simplifying the optimization procedure). So in comparison to other finite alphabet approaches we have improved symbol error rates with the pattern search algorithm.

Now, considering the other general approach for transmitting data through the compressed voice channel, which is mapping the data onto the parameters of a speech model, both [3] and [19] have found that this approach has higher error rates and higher computational complexity of the final modem operation when compared to the finite alphabet approach. Since the speech parameter method does not have symbol error rate per se with which we can compare our results, we instead need to look at bit error rates. Since most of these approaches include forward error correction coding of different degrees, we are mainly concerned with final coded data rate and bit error rate. For comparison we will look at alphabets with $N = 64$ and we use Reed–Solomon codes [43–45] as suggested in [3] with parameters chosen to be $M_{RS} = 63$, $K_{RS} = 45$, $T_{RS} = 18$. This means the FEC overhead is 30% and correcting capabilities were 15%.

The speech parameter modulation approach over the EFR codec was built on a CELP speech model in [4, 5, 15], and the raw uncoded bit error rate at 3 kb/s data rate was 2.9%. Adding error correcting codes (rate $\frac{1}{2}$ convolutional codes) yielded a throughput of 1.2 kb/s with 0.03% BER. Compare this to the 3 kb/s alphabet 64×16 which had an uncoded symbol error rate of 0.35% or uncoded bit error rate of 1.08%. After the FEC is added, the data rate was 2.1 kb/s and the bit error rate was reduced to less than 1×10^{-6}. So the uncoded bit error rate generated here is almost 3 times lower for the same data rate, and the final coded data rate is 1.75 times higher with a bit error rate 2 order of magnitudes lower. A similar method using an autoregressive speech model is proposed [16], transmitting data through the EFR vocoder at a data rate of 533 b/s, and a raw bit error rate of 2.6%. Compare this to the lowest data rate alphabet generated here (64×64) at 750 b/s with a symbol error rate and bit error rate of less than 10^{-6}. So through the EFR vocoder the method proposed here clearly outperforms the published results of the speech parameter approach.

The speech model parameter approach has also been applied to the FR vocoder [17, 18]. These studies also map data onto the speech parameters including phases and pitch frequencies related to formants in a waveform produced by an autoregressive speech model. In [17] a raw throughput of 2.0 kb/s was reduced to 1.15 kb/s with 0.02% bit error rate when coded with a punctured $\frac{1}{2}$-rate convolutional code. When we apply the RS codes to the 64×24 (2 kb/s) alphabet the result is a final data rate of 1.4 kb/s and a coded bit error rate of 4×10^{-6}, that is a higher data rate and a bit error rate almost 2 orders of magnitude lower. The same method over the FR codec in [18] produced a bit error rate 10^{-5} after error correction coding, and a data rate of 800 bps. This is also a higher error rate and lower data rate than achieved over the FR codec with the finite alphabet approach and Reed–Solomon codes.

Thus for both the EFR and FR codecs, the finite alphabet method presented here outperforms the speech parameter approach in terms of lower bit error rates and/or higher data rates after error correcting coding. Although the speech parameter modulation does reduce or eliminate the need for the initial optimization required by finite alphabets, this does not seem a significant advantage since this optimization is only required once and can be performed offline before the modem goes into operation. As the pattern search optimization also improved on the performance of the finite alphabets generated by previous studies, we can conclude that to the best of the authors' knowledge the modem outlined here has the best performance thus far published for data communication over compressed voice channels.

5 Discussion and Conclusion

In this paper we have shown that data transmission is possible through four different vocoders with the optimized finite alphabet approach, in addition to the GSM EFR vocoder considered in our previous study. In fact, data transmission is possible through all of the vocoders used in the GSM network (EFR, HR and AMR), showing that the proposed modem could work through any configuration of the widely used GSM network (and by extension through a UMTS network which uses the AMR vocoder as well). Similarly, it was shown that data transmission is also possible through the G729 voice codec commonly used for VoIP applications. The applicability of the method to all of these voice codecs is assumed to be due to the fact that all use different forms of CELP coding as the basis for speech compression. One big difference between the codecs is the degree of compression applied to the speech signal, and it was seen that the data rates achieved through the various codecs increased with the vocoder bit rate, that is to say that the data rate for our data-over-voice modem decreased with higher compression levels, as would be expected.

This raises the question of how the data rate (and hence the size of the finite alphabet) should be chosen for a network like GSM where a range of codecs, and/or a variable compression rate codec could be used by the network. In this situation three possibilities spring to mind. Firstly, the mobile units may be capable of requesting a particular voice codec from the network (for our case, this is equivalent to rate negotiation), and this would allow a finite alphabet to be chosen that was optimized for the voice codec requested. Secondly, some of the modern mobile units can report which vocoder type is used to compress the current call. In this case, the system could choose an appropriate FA from the set of available alphabets. Thirdly, for mobile units which do not support requesting a certain codec, it may be possible to use an adaptive communication system where the modem can detect the properties of the compressed voice channel and choose the appropriate finite alphabet from among a number that it has available—in negotiation with the modem on the other side of the communication, of course. The precise algorithm to do this is beyond the scope of this paper, but it could be as simple as probing with various finite alphabets (each optimized for the various possible codecs) until one is found to operate within reasonable error rates.

The other main innovation of this study, apart from showing that data transmission is possible over other voice codecs at reasonable error rates, is the introduction of the pattern search optimization technique for the purpose of finite alphabet generation. This optimization method was found to give significant benefits over the cooperative genetic algorithm method presented in [3], both in terms of convergence rate and properties (exponential convergence as opposed to roughly linear convergence) and final performance. For example in the case of the 4 kb/s data rate finite alphabet over the EFR vocoder, the pattern search algorithm converged to a symbol error rate about 16% lower than cooperative genetic algorithm. This difference in optimization performance is thought to be due firstly to the fact that the pattern search algorithm described in this paper is optimizing the finite alphabet as a whole, while the cooperative genetic algorithm used in [3] was optimizing the fitness of the individual symbols and relying on niching to generate an alphabet that functioned well when all the symbols were combined (this is the "cooperative" part of the cooperative genetic algorithm). Since total alphabet performance is what is finally important, it is not surprising that using total alphabet performance as the cost function produces better results than only indirectly optimizing this performance through individual symbol error rates. Secondly, final convergence of the pattern search algorithm is achieved through the reduction of the step size, whereas final convergence of the cooperative genetic algorithm was achieved through the reduction of the number of offspring per iteration, and while the former can be finely adjusted down to arbitrarily small values, the latter is an integer and so has a degree of granularity, giving generally worse convergence.

While a review of the data rates possible through various codecs (see Fig. 11) shows that data transmission through the compressed voice channel will not produce high transmission rates for multimedia downloads, the data rates of several kilobits per second would be sufficient for common machine-to-machine applications such as vehicle telemetry, encrypted voice communications, smart meters, alarm systems, security systems, and financial applications like wireless automatic teller machines (ATM) and point-of-sale (POS). For these types of applications the use of a compressed voice channel, especially the GSM voice channel, would give an advantage of very wide coverage due to the geographical footprint of the legacy 2G voice networks, as well as a high level of reliability in congested urban areas due to the higher priority usually given to voice traffic of packet switched data and the fact that the call is circuit switched so once a call is connected the channel will remain available until released.

The method of modem design described here—using pattern search optimization to create a finite alphabet that is decoded by a maximum likelihood estimator—could be used to communicate over channels other than compressed voice networks with appropriate modification of the cost function, and the authors envision that the future work on this subject involves the application of the concept to other channel types.

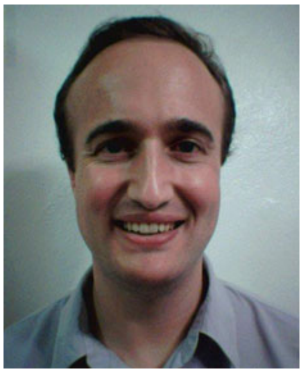
References

1. Digital cellular telecommunications system (Phase 2+) (1998). Physical layer on the radio path; general description; (GSM 05.01 Version 7.1.0. Release 1998), Available: ETSI TS 100 573.
2. GSM World statistics GSM Association (2007). Available: http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm. Accessed 2008.
3. LaDue, C., Sapozhnykov, V., & Fienberg, K. (2008). A data modem for GSM voice channel. *IEEE Transactions on Vehicular Technology, 57*(4), 2205–2218.
4. Katugampala, N. N., Al-Naimi, K. T., Villette, S., & Kondoz, A. M. (2004). Real time data transmission over GSM voice channel for secure voice and data applications. *Secure mobile communications forum: Exploring the technical challenges in secure GSM and WLAN, 2004* (pp. 7/1–7/4). The 2nd IEE (Ref. No. 2004/10660) 23 September 2004.
5. Katugampala, N. N., Villette, S., & Kondoz, A. M. (2003). *Secure voice over GSM and other low bit rate systems*. IEE secure GSM and beyond: End to end security for mobile communications, London.
6. 3GPP TS 43.064 (2005). Overall description of the GPRS radio interface, Stage 2.
7. Ni, S., & Haggman, S. (1999). GPRS performance estimation in GSM circuit switched services and GPRS shared resource systems. In *Proc. IEEE WCNC* 1999 (pp. 1417–1421).
8. Alanko, T., Kojo, M., Laamanen, H., Lilijeberg, M., Moilanen, M., & Raatikainen, K. (1994). *Measured performance of data transmission over cellular networks*. University of Helsinki, Department of Computer Science, Finland, Report C-1994-53.
9. Liu, H.-H., & Wu, J. L. C. (2002). Delay analysis of integrated voice and data service for GPRS. *IEEE Communications Letters, 6*(8), 319–321.
10. Madkour, M. F. (2003). Effect of high GSM voice traffic on GPRS data network and the proposed solutions. In *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, 2003. MWSCAS '03* (Vol. 3, pp. 27–30, 1259–1262).
11. Halonen, T., Romero, J., & Melero, J. (2003). *GSM, GPRS and EDGE performance* (2nd ed.). England: John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ.
12. Hamiti, S., Hakaste, M., Moisio, M., Nefedov, N., Nikula, E., & Vilpponen, H. (1999). EDGE circuit switched data-an enhancement of GSM data services. *Wireless Communications and Networking Conference, 1999. WCNC. 1999* (pp. 1437–1441). IEEE.
13. Mullner, R., Ball, C.F., Ivanov, K., Treml, F., & Spring, G. (2004). Quality of service in GPRS/EDGE mobile radio networks. *IEEE 59th, Vehicular Technology Conference, 2004* (Vol. 5, pp. 2507–2511).
14. Goetz, I. (2003). Keeping up with GPRS. *Communications Engineer, 1*(2), 46–46.
15. Katugampala, N. N., Al-Naimi, K. T., Villette, S., & Kondoz, A. M. (2005). Real time end to end secure voice communications over gsm voice channel. In *13th European signal processing conference*. Antalya, Turkey.
16. Kotnik, Z. Mezgeca, J. Svecko, & Chowdhury, A. (2009). Data transmission over GSM voice channel using digital modulation technique based on autoregressive modeling of speech production. *Digital Signal Processing, 19*(4), 612–627.
17. Rashidi, M., Sayadiyan, A., & Mowlaee, P. (2008). Data mapping onto speech-like signal to transmission over the GSM voice channel. In *Proc. 40th Southeastern symposium. system theory* , (pp. 54–58). New Orleans, LA, 16–18 March 2008.
18. Rashidi, M., Sayadiyan, A., & Mowlaee, P. (2008). A harmonic approach to data transmission over GSM voice channel. In *Proc. 3th IEEE int. conf. information and communication technologies: From theory to applications* (pp. 1–4). 7–11 April 2008.
19. Shahbazi, A., Rezaie, A. H., Sayadiyan, A., & Mosayyebpour, S. (2009). A novel speech-like symbol design for data transmission through GSM voice channel (pp. 478–483).
20. Digital cellular telecommunications system (Phase 2) (200) Enhanced Full Rate (EFR) speech transcoding (GSM 06.60 version 4.1.1). European Telecommunications Standards Institute. Available: ETSI EN 301 245 V4.1.1 (2000–2008).
21. Redl, S., Weber, M., & Oliphant, M. W. (1998). *GSM and personal communications handbook*. Arctech House Inc, 685 Canton Street, Norwood, MA 02062.
22. Digital cellular telecommunications system (Phase 2+) (2000). Half rate speech; Half rate speech transcoding (GSM 06.20 version 8.0.1 Release 1999). European Telecommunications Standards Institute. Available: ETSI EN 300 969 V8.0.1 (2000—2011).
23. European digital cellular telecommunications system (Phase 2) (1997) Full rate speech processing functions (GSM 06.01 version 5.0.1). European Telecommunications Standards Institute. Available: ETSI ETS 300 960.
24. Digital cellular telecommunications system (Phase 2+); Adaptive Multi-Rate (AMR) speech transcoding (GSM 06.90 version 7.2.1 Release 1998). European Telecommunications

Standards Institute, 2000. Available: ETSI EN 301 704 V7.2.1 (2000–2004).
25. Meston, R. (2003). Sorting Through GSM Codecs: A Tutorial, Racal Instruments, Irvine, CA. Available: http://www.commsdesign.com/showArticle.jhtml?articleID=16501605.
26. Coding of speech at 8 kbit/s using conjugate structure algebraic-code-excited linear prediction (CS-ACELP) (2007) International Telecommunication Union. Available: http://www.itu.int/rec/T-REC-G.729-200701-I/en.
27. Manolakis, D. G., Ingle, V. K., & Kogon, S. M. (2000). *Statistical and adaptive signal processing*. New York: McGraw-Hill.
28. Haykin, S. (2001). *Communication systems* (4th ed.). New York: John Wiley & Sons.
29. Proakis, J. (2001). *Digital communications* (4th ed.). New York: McGraw-Hill.
30. Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization, 7*(1), 1–25.
31. Lewis, R., & Torczon, V. (1999). Pattern search methods for bound constrained minimization. *SIAM Journal on Optimization, 9*(4), 1082–1099.
32. Lewis, R., & Torczon, V. (2000). Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization, 10*(3), 917–941.
33. Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.). Prentice Hall.
34. Schroeder, M. R., & Atal, B. S. (1985). Code Exited Linear Prediction (CELP): High quality speech at very low bit rates. In *Proc. ICASSP'85* (pp. 937–940).
35. Kolda, T., & Torczon, V. (2004). On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization, 14*(4), 939–964.
36. Dennis, J. Jr., & Torczon, V. (1991). Direct search methods on parallel machines. *SIAM Journal on Optimization, 1*(4), 448–474.
37. Hough, P., Kolda, T., & Torczon, V. (2001). Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing, 23*(1), 134–156.
38. Lewis, R., Shepherd, A., & Torczon, V. (2007). Implementing generating set search methods for linearly constrained minimization. *SIAM Journal on Scientific Computing, 29*(6), 2507–2530.
39. ETSI EN 300 724 (2000). Digital cellular telecommunications systems (Phase 2+); ANSI-C code for the GSM Enhanced Full Rate (EFR) speech codec; (GSM 06.53 version 8.0.1. Release 1999). European Telecommunications Standards Institute.
40. Phillips, C. L., & Parr, J. M. (1995). *Signals, systems and transforms*. Upper Saddle River: Prentice-Hall.
41. Hanzo, L., Münster, M., Choi, B. J., & Keller, T. (2003). *OFDM and MC-CDMA for broadband multi-user communications, WLANs and broadcasting*. Chichester: John Wiley & Sons.
42. The Tesla C870 GPU computing processor. Available: http://www.nvidia.com/object/tesla_c870.html. Accessed 2008.
43. Lin, S., & Costello, D. J. Jr. (1983). *Error control coding: Fundamentals and applications*. Englewood Cliffs: Prentice-Hall.
44. Moon, T. K. (2005). *Error correction codes: Mathematical methods and algorithms*. Hoboken: John Wiley & Sons.
45. Wicker, S. B. (1995). *Error control systems for digital commuications and storage*. Upper Saddle River: Prentice Hall.

Vitaliy V. Sapozhnykov was born in Kharkov, Ukraine (former USSR) on January 4, 1970. He received the B.S.E. degree (with First Class Honors, Gold Medal Award) and the Ph.D. degree from the Kharkov University, Ukraine, both in electrical engineering, in 1992 and 1996 respectively.

Until year 2009 he held senior R&D positions for various companies developing advanced communication technologies. Dr Sapozhnykov is currently a principal DSP engineer with Ofidium Ltd, Melbourne, Australia. His research interests span the broad area of digital communications, pattern recognition, adaptation and stochastic optimization.



Kurt S. Fienberg was born in Gosford, Australia, in 1976. He received the degree of B. Sc. (with First Class Honours) in 1998 with majors in physics and mathematics, and was awarded the Ph.D. degree in remote sensing and climatology in 2003, both from the University of Tasmania in Hobart, Australia. From 2004 to 2006 he worked in algorithm development and signal processing for telecommunication technologies at Symstream Technology Group Ltd. He is currently Senior DSP Engineer at Dynamic Hearing Pty Ltd in Melbourne Australia.

His research interests cover a range of areas, including remote sensing, radiative transfer, signal processing, adaptive systems and stochastic optimization. He is also interested in scale invariance and multi-scale processes in geophysical fields, including clouds, rainfall, sediment transport, and turbulence, and the effects that the high variability and intermittency of these fields have on remote sensing and modeling.