Q-ary Repeat-Accumulate Codes for Weak Signals Communications

Nico Palermo – IV3NWV (e-mail: nicopal@microtelecom.it)

Introduction

Whether coherent or incoherent signal detection is used, channel coding systems designed for amateur radio weak signal communications are currently based on strong, large constraint length convolutional codes or on Reed-Solomon codes.
The WSJT-X program [1], in example, offers four basic modes, namely JT4, JT9, JT65 and WSPR which use incoherent demodulation over an orthogonal MFSK signal set and a binary, k=32 constraint length, R=1/2 rate convolutional coding (JT4, JT9 and WSPR) with a moderate size signal constellation (4-FSK or 8-FSK), or a Reed-Solomon code and a larger 64-FSK signal constellation (JT65).
EMEpsk [2] instead attempts to estimate the channel state and coherently demodulate a BPSK signal using a large constraint length binary convolutional code.
While they are someway remarkably good, both convolutional codes and Reed-Solomon codes do not provide a performance which is very close to the limits predicted by information theory. This fact motivated a continuous research which in the '90s lead to the discovery of turbo codes by Berrou [3] , the rediscovery of the Gallager's LDPC (Low Density Parity Check) codes and the message passing decoding algorithm devised by David MacKay [4].
Here we present a particular class of LDPC codes named QRA, shortly for Q-ary irregular Repeat-Accumulate codes, which exhibit a performance which is closer to the limits predicted by theory. Despite their simplicity, QRA codes appear to be as good as the best known LDPC codes and, unlike them, can be encoded in a linear time [5]. Being non-binary, QRA codes are perfectly suited for a direct implementation on MFSK communication systems and offer a significant coding gain over conventional coded modulation systems based on convolutional or Reed-Solomon codes.
In Section I of this paper, 'QRA Code Basics', we will introduce QRA codes, how they can be concisely described by a graph and some heuristic rules which should help to design or at least to avoid selecting the worst of them.
Section II will present the results of the simulations and compare the coding system used by JT65 (RS(12,63) code over GF(64) + Koetter-Vardy RS soft-decoder) vs. a QRA(12,63) code over the same alphabet and a Q-ary MAPP (Maximum A Posteriori Probability) message passing decoder.
In Section III, 'Message Sequences Aided Decoding', we will explore in more detail the semantic and the redundancy of messages exchanged by two communication nodes (stations) in order to validate with sufficiently low uncertainty the information they have to exchange, as it normally happens when a 2-way connection (QSO) needs to be established and confirmed. We will show that in such a 2-way connection the correctly decoded messages received up to a particular connection phase can be used by a MAPP soft-decoder as a priori knowledge information and improve the receiver detection threshold as much as shorthands and low informative messages do, yet carrying full information, unlike the JT65 protocol shorthand messages, about the sender and recipient addresses.
Unlike the deep search algorithm used in JT65 which relies on a database of communication nodes or other information considered to be plausible, a MAPP decoder can improve its message detection sensitivity much just by exploiting the few messages it is receiving during a specific QSO. This decoding method is indeed inferior to a deep-search based message decoding, nonetheless 1) it provides a significant advantage over a system in which the semantic of the QSO is ignored and 2) it relies only on information received from the communication channel.

I - QRA Codes Basics

QRA codes are very simple codes.
In the most simple form of the encoding process, the K information symbols x_1, x_2,... x_K, each of them belonging to a finite field of size Q (the Galois field GF(Q)), are repeated accordingly to a repetition factor denoted with r_k, and produce the sequence of symbols:

$$\underbrace{x_1\ x_1 \ldots x_1}_{\text{repeat } r_1 \text{ times}}\quad \underbrace{x_2\ x_2 \ldots x_2}_{\text{repeat } r_2 \text{ times}}\quad \underbrace{x_3\ x_3 \ldots x_3}_{\text{repeat } r_3 \text{ times}}\quad \ldots\quad \underbrace{x_K\ x_K\ x_K \ldots x_K}_{\text{repeat } r_K \text{ times}}$$

The repetition factors r_k don't need to be equal. In the case they are the code is called a regular code (all the information symbols are repeated by the same factor); if they don't, the code is called an irregular code (being the irregularity in the differences of the repeat factors used for each information symbol).

Example: consider a sequence of K=4 symbols {A, b, f, J} which is the information we want to send. We decide to repeat the first symbol 'A' three times (r_1=3), the second and the third symbols 'b' and 'f' five times (r_2=r_3=5) and the last symbol 'J' six times (r_4=6).
The repeated sequence of symbols will be therefore : AAAbbbbbfffffJJJJJJ, and the sequence r_1=3, r_2=5, r_3=5, r_4=6 is the sequence of the repeat factors.

The sequence of repeated information symbols is then reordered accordingly to a given permutation sequence Π and produce as output the sequence of symbols:

$$x_{\pi 1}\ x_{\pi 2}\ x_{\pi 3}\ x_{\pi 4}\ x_{\pi 5}\ x_{\pi 6} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ x_{\pi M}$$

where $M = \sum_{n=1}^{K} r_n$ is the length of the sequence of the repeated information symbols.
The M parity checks of the code are generated from the above sequence using a recursive formula which generates a sequence of check symbols y_m as a 'weighted' accumulation of the input sequence $x_{\pi 1}$:..$x_{\pi M}$:

$$\begin{aligned} y_0 &= 0; \\ y_m &= y_{m-1} + w_m \cdot x_{\pi m} \\ m &= 1,2,3,...,M \end{aligned}$$

where w_m are non-null multiplicative factors, called weights, taken from the GF(Q) field. The '+' and the ' · ' operators are not the normal algebraic operators but the additive and the multiplicative operators on the GF(Q) field (see Appendix A).
The encoder output codeword $\underline{x}$ is formed concatenating the sequence of the original K information symbols x_1...x_K with the sequence of the M parity check symbols y_1...y_M computed by the accumulator obtaining a K+M = N symbols sequence:

$$\underline{x} = (x_1, x_2, \ldots x_N) = (x_1, x_2, \ldots x_K, y_1, y_2, \ldots y_M)$$

The rate of RA codes constructed in this way is R = K/(K+M) and since $M = \sum_{k=1}^{K} r_k$ is always

larger than than K when $r_k>1$, the code rate can't be larger than 1/2 and, for practical reasons, it's usually much smaller (R<1/4).
To construct RA codes with a larger rate the sequence of symbols $x_{\pi 1}:..x_{\pi M}$ is grouped in groups of *a>1* symbols per group (*a* is named the grouping factor of the code), all the symbols in a group are xor-ed together and only a parity check per group is generated. In this case the code rate is R = K/(K+M/a) and thus can be higher than the rate of a code in which no symbol grouping is used.
All the QRA codes constructed in this way are called systematic codes (their codewords contain an explicit copy of the source information symbols) and, as we will see later, they are particularly useful in the case that some information symbols are known in advance by the decoder.
As any other linear code, QRA codes may be described by a graph which, besides being extremely important for decoding purposes, provides an immediate sight of the underlying code structure.
For this purpose we denote each of the N symbols x_n of a codeword with a circle (named a variable node) and each of the M check equations $y_m = y_{m-1} + w_m \cdot x_{\pi m}$ with a box (named a check node). Then we connect circles to boxes with edges to indicate which code variables are involved in which code checks. The resulting graph is named a Tanner graph [6] and for an irregular QRA code it looks like in Fig. 1.

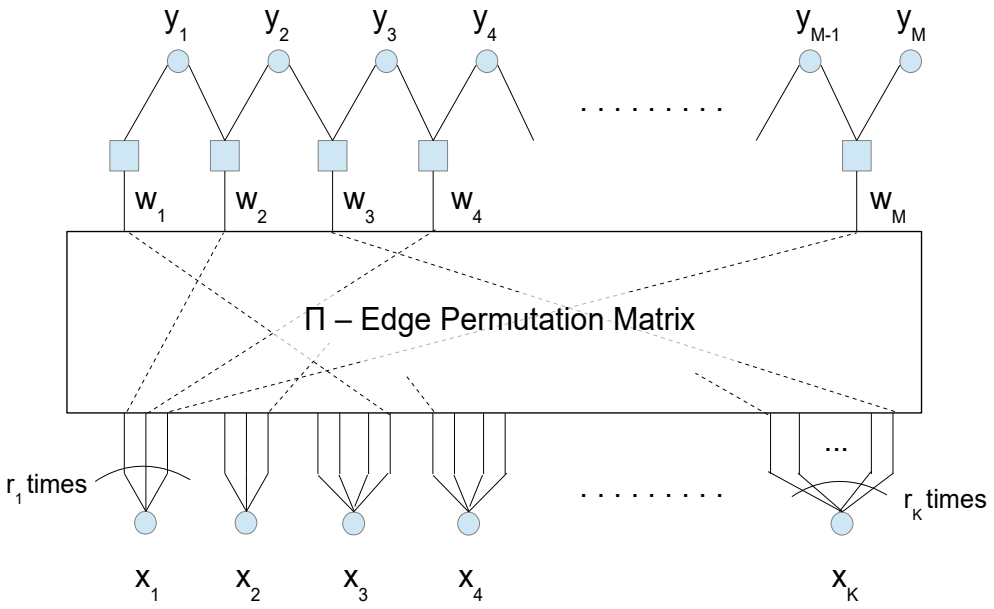

Fig. 1 – The Tanner graph of an irregular QRA Code with grouping factor a=1. The circles $x_1..x_K$ are the variable nodes associated with the (systematic) information symbols, the circles $y_1..y_M$ are the variable nodes associated with the parity check symbols and the boxes are the check nodes which represent the code parity equations.

From Fig.1 we can see that the variable degrees, that's to say the number of check nodes a variable node is connected to is no more than $\max(r_k)$ and that the check nodes degree, that's to say the number of variables connected to a check node, is not larger than three. So far, if we described this code with a MxN parity check matrix H, the matrix for which each codeword of the code satisfies the set of M equations $H\underline{x} = 0$) each of its row would have no more than three non-null entries (the

maximum check degree) and each of its columns would have no more than max(r_k) non-null entries (the maximum variable degree), being the average value lower than this maximum.
The parity check matrix of a QRA code is therefore very sparse (few non-null entries per row and few non null entries per column) and suggests that both regular and irregular QRA codes are, by definition, Low Density Parity Check codes indeed!
From Fig. 1 we see also that an irregular QRA code can be designed with some freedom degrees and that it is completely described by three sets of variables:

- the set of the repetition factors $r_1...r_K$ and the grouping factor *a*,
- the edge permutation matrix Π which permutes the repetitions of the information symbols,
- the parity nodes weights $w_1...w_M$

Therefore, which choice of the sequence of repetition factors, grouping factor, edge permutation matrix and code weights leads to the code which, once decoded, offers the lowest word error rate at a given Eb/No ratio? Unlike convolutional and RS codes, QRA codes and LDPC codes lack a true algebraic framework and their construction is often based on a random Montecarlo approach in which a candidate code with some desired properties is simulated, its word error rate curve computed and eventually selected if its performance is better than the average.
The Tanner graph of the code, along with what we know about the message passing decoding, gives some hints on the codes we should avoid. Such hints help to select a 'better than the average' code in a class but the problem is still open, at least for short codes.
So far, how to design and select the proper repetition factors, the edge permutation matrix and the code check weights (a further freedom degree that binary RA codes have not to deal with) when the code is made by very few information symbols?
I (partially) solved this problem with few selection guidelines which helped me finding codes which looks not very bad and at least outperform the Reed-Solomon code used in JT65 quite abundantly.
These rules are rather intuitive and they are listed here below:

a. Systematic symbols repetition factors:
   - irregular codes are always better than regular codes, repetition factors should be properly selected from the set of the integer numbers {3..10},
   - repetition factors lower than three lead to codes with low codeword weights which can be likely confused in a short code decoder and should be avoided,

b. Edge permutation matrix:
   - cycles in the resulting code Tanner graph should be as large as possible. Fig 1, in example, shows that there's a cycle of six edges which touches the variables x_1, y_2 and y_3. Short cycles limit the ability of a message passing decoder to compute exact probabilities and should be avoided.
   - the edge permutation matrix must be designed so that short length cycles are avoided. The minimum cycle length in a code is called 'girth'. Low code girths lead to undesirable information correlation when the code is decoded iteratively by the message passing algorithm. Design the edge permutation matrix so that the code girth is at least 6.

c. Check node weights:
   - weights should be selected in order to maximize codewords distance. In binary codes weights can't be different from 1 (the only non null element of GF(2)), but with Q-ary codes we have a further freedom degree which helps to keep such a distance larger than that of any binary code.

- the weights which refer to the same information symbol (as i.e. w_1, w_4 and w_M which weight the information symbol x_1 as shown in Fig. 1) must be properly selected. In the QRA codes presented in this paper we choose these weights so that for any information symbol x_j the associated weights $w_k(j)$, with k=1..r_j obey the rules:

  1) $\sum_{k=1}^{z} w_k(j) \neq 0$ for any z<r_j, and

  2) $\sum_{k=1}^{r_j} w_k(j) = 0$

While someway arbitrary the weighting rules grant that the accumulator of the code is terminated to zero for any codeword in the code. The consequence of this choice is not trivial. It can be demonstrated that under the framework of iterative, turbo-like decoding and as shown in [7], the EXIT chart [8] of the constituent accumulator sub-code have the property that the extrinsic information $I_{ex}(I_a)$ provided by the accumulator decoder when it is fed by perfect a priori knowledge (I_a=1) is $I_{ex}(1)$=1 for any input SNR. This property boosts the ability to successfully decode the received signal whenever a sufficient amount of a priori knowledge about the transmitted codeword is available to the decoder.

II - Simulations Results.

To compare the performance of a QRA code and decoder with the performance of some already well known coding scheme, I searched for some irregular QRA codes with the same parameters of the Reed-Solomon code used in Joe Taylor's JT65 mode, that's to say a code with K=12 information symbols and a codeword length of N=63 symbols taken from an alphabet of Q=64 symbols. To accomplish this task I wrote an algorithm which implements the code design guidelines mentioned in section I and obtained some few codes whose performance I then evaluated with a MAPP soft-decoder for Q-ary LDPC codes based on the message passing algorithm. A description of this decoder is available in Appendix B.

Performance simulations have been made for two type of channel models, the classical AWGN (Additive White Gaussian Noise) model, and the flat block-fading Rayleigh model. For this last model we assume that the channel is affected by a frequency flat fading and that the N multiplicative channel gains a_n which affect the received symbol amplitude are normally distributed complex variables, uncorrelated from symbol to symbol, with average squared amplitude $\overline{a_n^2} = 1$ for each of the n=1..N codeword symbols.

Fig. 2 shows the Word Error Rate of the 64-ary QRA(12,63) code/decoder in a AWGN channel and compares it with the code/decoders performance used in JT65. The curve of the QRA code is labeled 'QRA' and was obtained allowing the decoder to iterate for a maximum of 100 iterations. The curves labeled FT, KV and BM refers respectively to the Franke-Taylor soft-decoder, to the patented Koetter-Vardy decoder used in earlier implementations of JT65 and to the Berlekamp-Massey hard-decision decoder as published in [10].

The QRA code performance shown in Fig. 2 is referred to the code QRA(12,63) for the 64-FSK modulation whose parameters are listed in Appendix A. On the AWGN channel this code offers a gain of about 1.3 dB over a RS(12,63) Reed-Solomon decoded by the FT decoder and about 1.6 dB over the KV decoder.

The code performance is remarkable, especially in consideration that the theoretical minimum Eb/No ratio at which a 64-FSK system with incoherent demodulation using a code rate R=12/63 (and infinitely long codewords) can communicate with an arbitrary small error rate in a AWGN

channel is about 2.1 dB [11]. This is known as the Shannon Limit or the channel capacity. The QRA(12,63) coding system Eb/No threshold, arbitrarily taken at a 50% word error rate, is about 2.7 dB and only 0.6 dB far away the channel capacity. Note also that, at practical word error rates of few percents, the distance of the QRA decoder from the Shannon limit is half the KV decoder distance.

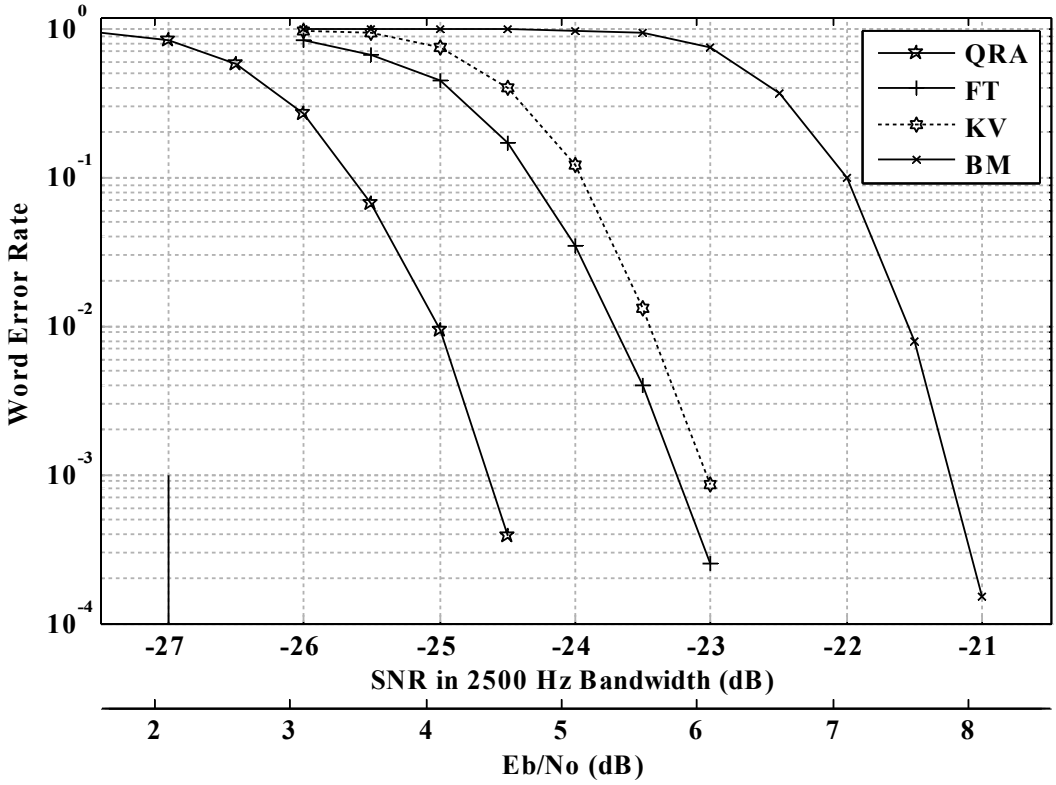

Fig. 2 – Word Error Rate of the QRA(12,63) code with the message passing decoder on the AWGN channel compared with the RS(12,63) Reed-Solomon code decoded by the Franke-Taylor (FT), the Koetter-Vardy (KV) and the hard-decision Berlekamp-Massey (BM) decoders used in JT65. The vertical bar at Eb/No = 2.1 dB is the Shannon capacity limit of a 64-FSK channel with incoherent demodulation and code rate R=12/63.

The horizontal axis scales used in Fig. 2 are the Signal to Noise Ratio in a 2500 Hz reference bandwidth and the Eb/No ratio. The relationship between the SNR in a given bandwidth and the Eb/No ratio depends on the Eb/No ratio through the system effective communication bit rate Rb, the reference bandwidth B and the ratio between the symbols used for synchronization purposes (Nsync) and the information-carrying data symbols (N) accordingly to the formula:

$$SNR = \frac{Rb}{B}\left(1+\frac{Nsync}{N}\right)\frac{Eb}{No}$$

In JT65, for instance, we have Rb = 72 bit/46.81 s = 1.538 bit/s, Nsync=63 and N=63. Therefore, once expressed in dB, the SNR scale is offset by -29.1 dB respect to the Eb/No scale.
Note that, as the performance of a coding system in terms of the Signal to Noise Ratio in an arbitrary noise bandwidth depends on the effective communication rate Rb and the Nsync/N ratio, the SNR scale is useful to compare different coding systems only when they operate at exactly the

same rate and with the same pilot/data symbols ratio. Nonetheless, as long as a coding system performance is evaluated in an AWGN channel, the communication rate is totally irrelevant. When expressed as a function of the Eb/No ratio the word error rate of a coding system is exactly the same whether the communication rate is 1 Bit/s or 1 GBit/s. Of course, since the energy spent per transmitted bit is a product between power and time, the faster the communication rate is, the shorter the bit time interval and the higher the power required to achieve the same error rate of a slower system.

Fig. 3 shows the QRA(12,63) message passing decoder performance as a function of the maximum number of iterations it is allowed to run. Ten decoder iterations are not sufficient to propagate all the available information to the code graph nodes and the decoder performance, while still better than the KV decoder, is less than optimal. This happens because with the message passing algorithm it takes k iterations for any information message to propagate and reach a destination node which is 2*k edges far away from the information source node. The QRA(12,63) code accumulator chain is ~100 edges long and it takes fifty iterations for a message originated by the first accumulator symbol to reach the last one. Nonetheless, twenty decoder iterations are sufficient to get out of the code almost all of its coding power and the full coding gain is achieved with 100 iterations. The effective decoding time has a small, less than linear dependence on the maximum number of the iterations as most of the successful decodes occur within a number of iteration which is usually less than the allowed maximum.

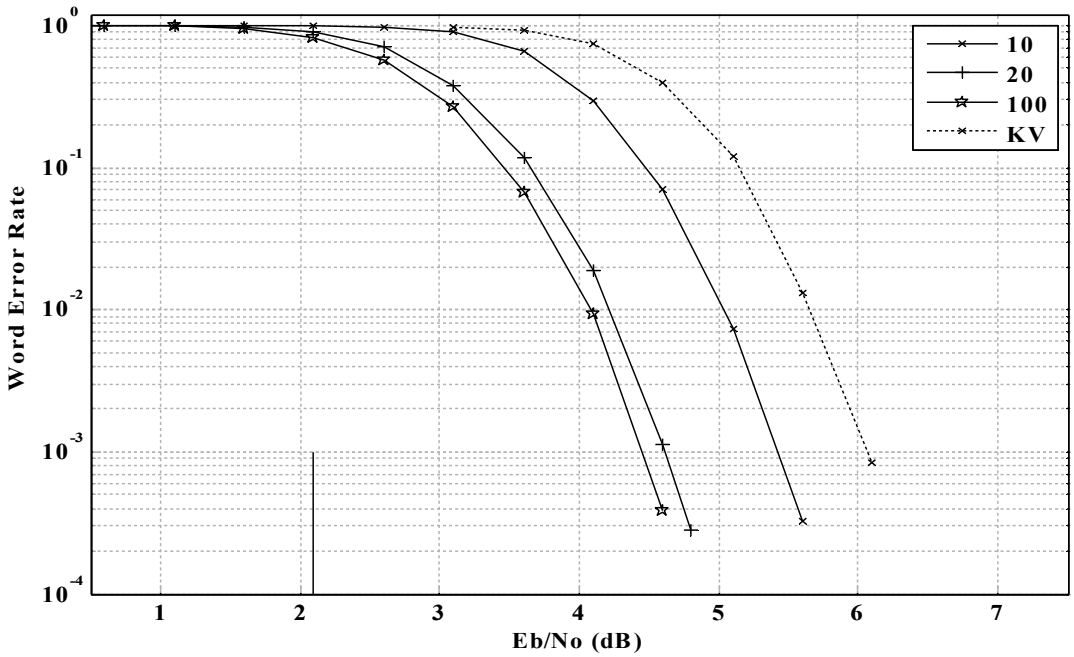

Fig. 3 – QRA(12,63) code performance as a function of the maximum number of iterations the message passing decoder is allowed to run. The KV RS(12,63) Reed-Solomon soft-decoder performance curve is plotted for reference. The vertical bar at Eb/No = 2.1 dB is the Shannon capacity limit of a 64-FSK channel with incoherent demodulation and code rate R=12/63.

Simulations results for the Rayleigh block-fading channel are shown in Fig. 4. The percentage copy curve of the QRA(12,63) code with the message passing decoder is labeled with QRA. The performance of the RS(12,63) code used by JT65 and decoded with the Franke-Taylor decoder is labeled with FT. The simulation of the QRA code assumes that no word is lost in the synchronization process and that the energy losses due to impairments in the synchronization

process are negligible. A maximum of 100 iterations are used in the message passing decoder. The curves labeled DS, FT and BM are those published in [10].
We see that for the Rayleigh channel the coding-gain of the message passing decoder versus the FT decoder is 1.5 dB, thus even higher than that achieved in the AWGN channel, and almost independent on the percent copy.

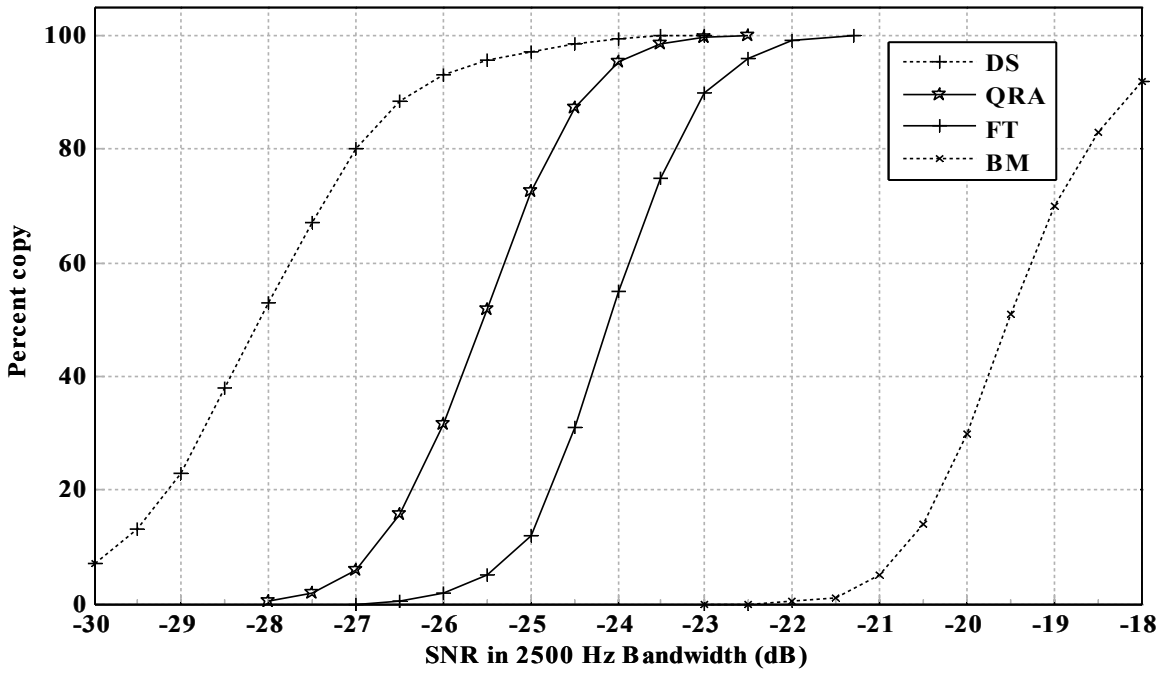

Fig. 4 – Percent copy for the QRA(12,63) code and message passing decoding versus the RS(12,63) code and FT decoding for the Rayleigh channel. The JT65 Deep-Search (DS) and BM decoders curves added for reference.

It should be observed, anyway, that while the QRA curve refers to a "block-fading" channel model, the FT curve refers to a different "1 Hz Doppler-spread" fading channel model and that a direct and more accurate comparison should take into account the differences between them.
It is also important to note that any soft-decoder aimed to achieve the channel capacity needs very accurate channel observations and symbol reliabilities. The message passing decoder makes no exceptions to this rule and in all the simulations it is fed by a symbol reliability metric which models accurately what the theory asks for. As we know from the statistics of a MFSK signal with incoherent demodulation, symbols likelihoods are proportional to a non-linear function of the amplitudes of the received symbol. The exact relationship between the likelihoods of the received amplitudes vector $\underline{r}$ given that the transmitted symbol x was X_j is:

$$\text{Prob}\,(\underline{r} \mid x=X_j) = \quad \Lambda j(\underline{r}) \propto I_0(r_j \sqrt{Es}/\sigma^2) \quad ,$$

where r_j is the j-th component of the received amplitudes vector, Es is the symbol energy, σ^2 is the noise variance per signal dimension ($\sigma^2 = No/2$) and $I_0(.)$ is the modified Bessel function of the first kind of order zero.
Therefore, to exactly evaluate the likelihoods from the received amplitudes we need to know both the noise variance and the symbol energy Es. In a 64-FSK system with a codeword length of some

tens of symbols the noise variance can be evaluated with a good approximation but, since the codeword is not sufficiently long, the estimation of the symbols energy can be affected by large errors that do not allow to compute the required likelihoods with sufficient precision. To overcome this problem the decoder is fed with likelihoods assuming that the symbols energies are those which we would receive at the decoding threshold of the code. This choice is of course optimal only when the received signal is right at the decoding threshold and for the AWGN channel only.

III – Message Sequences Aided Decoding.

Consider a minimal QSO between the two EME stations IV3NWV and SM5BSZ.
The sequence of messages exchanged by them may look as the following:

1 -	**CQ IV3NWV JN66 ...**	
2 -		**IV3NWV SM5BSZ JO89 ...**
3 -	**SM5BSZ IV3NWV -599 ...**	
4 -		**IV3NWV SM5BSZ -599 …**
5 -	**SM5BSZ IV3NWV 73s …**	
6 -		**IV3NWV SM5BSZ 73s …**

where the ellipsis symbol … indicates that, in the attempt to complete the QSO, when a reply from the destination station has not been copied a message can be eventually repeated by the sender.
As done in JT65 and in order to reduce to the essential the amount of the information which each message carries, we can encode any of these messages dedicating 28 bits to encode the call sign of the message source, 28 bits to the call sign of the message destination and 16 bits to encode Maidenhead locators, particular call sign prefixes or suffixes, signal reports or shorthand messages.
Despite this 'source coding' operation, we note that the sequence of messages exchanged in a QSO contain still a large amount of redundancy. The sender and destination call signs, for instance, are repeated over all the QSO.
In the example above, if SM5BSZ wishes to reply a call, he needs at least to wait for a call. In this case he does not need to listen to every possible message but just to those that begin with a CQ. Once he has received a CQ call, and after he replied to the IV3NWV call with message #2, he can found himself in two situations: 1) IV3NWV hasn't copied message #2 and continue to send a CQ with message #1 or, 2) IV3NWV has copied SM5BSZ's reply and is going to answer him with a report sent with message #3 in which both the addresses of the message are already known by the recipient.
This example shows that as long as the QSO proceeds to its conclusion, the stations which are participating to it need to receive less and less information from the other party. JT65 already exploits this opportunity in a simple way and, to improve the decoding ability of recipients, it sends the last messages of a QSO as handshakes, especially encoded messages which do not convey any address information.
A different decoding approach is to explicitly exploit the information received up to some phase of a QSO and use it as an 'a priori' knowledge which can be passed to the decoder in order to improve its sensitivity. Such a knowledge is passed to the QRA soft-decoder as a probability distribution. In example, if we assume that a symbol in a message is already known, we first pass to the decoder a probability distribution in which we set to zero all the distribution entries that we exclude as possible outcomes; then we let the decoder run and find the best estimation of the message symbols we still know nothing about. What we desire to be excluded from the decoding process is simply assigned a null probability. Following this approach I've simulated the performance of the decoder accordingly to the message fields that are already known by the receiver during the QSO

accordingly to the phase the QSO has reached.
In the first phase of the QSO (messages #1 and #2 of the example above) each of the two stations decoders are waiting for a message which is either a CQ call or a reply in which the destination address of a message is his own call-sign. SM5BSZ is waiting for a CQ, IV3NWV is waiting for a reply directed to himself. In both of these cases we can drive the decoder with an a priori knowledge of 28 bits, which is the size of the destination address field, and look for the decoding performance we obtain when such an amount of information is provided to the decoder.
In the second phase of the QSO (messages #3 and #4) both the stations already know that any further message in the QSO have both a specific destination and a source address. In this phase of the QSO, the two stations are actually exchanging their signal reports, and they can drive the decoder with an a priori knowledge of 56 bits, the amount of information contained in the first two fields of a JT65 message. Finally, in the last phase of the QSO, the two stations need only to send an acknowledgment message to confirm what received up to the second phase. We can continue to consider this acknowledgment as a message in which the third field of the JT65 protocol has the intended meaning. In this phase the decoder can be fed with an a priori knowledge of all of the constituent 72 bits of the message (both addresses are known, third field known). This is equivalent to decode real shorthand messages and we might expect to get out from the decoder the same performance obtained when they are used.
Note that, since in each phase of the QSO a station could receive the repetition of a message sent in a previous phase (this happens whenever the other party has not decoded our last reply and retries a transmission), the decoder must be occasionally run twice, once for any of the two messages that the receiver expects in a particular QSO phase. From a practical point of view, the decoder needs the intrinsic symbols information, that's to say the likelihoods computed from the channel observations; thus we can simply mask to zero all the likelihoods which we do not consider plausible.
In Fig. 5 we show the performance of a decoding system based on this approach when the QRA(12,63) code is decoded with some amount of known information and a block-fading Rayleigh channel model is assumed.
The curve labeled AP28 refers to the decoding of messages in which the first 28 bit field of the message is known to the receiver. This situation happens whenever we are waiting for a CQ call or for a a reply to our own call-sign (in this case the destination address of the reply message is our call sign, therefore we already know it in advance).
The curve labeled AP56 refers to the decoder performance when both the sender address and the destination address of the incoming message are known by the receiver, as it happens when the receiver is waiting for a signal report, an acknowledgment or some other shorthand information.
The sensitivity to CQ calls and replies directed to the caller (curve AP28) is 2.5 dB better than the sensitivity obtained with the RS code and the FT decoder currently used in JT65.
The sensitivity to signal reports and shorthand messages (curve AP56), which still carry 16 bits of unknown information, is remarkably superior to that of the deep-search algorithm. This is not really surprising, anyway. The deep-search is a maximum likelihood algorithm which looks for the most probable codeword among a restricted set of candidates; the QRA decoder computes symbol-wise probabilities which, being marginal distributions (or weighted averages) of all the possible cases, produce a better estimate of the transmitted symbol messages. Such a better sensitivity suggests that a more uniform content among messages could help lowering the decoding threshold even more. The Maidenhead locator, in example, could be transmitted before the end of a QSO and prior to the final acknowledgments. In this case the CQ/reply messages in the initial phase of the QSO would contain only 28 bit of information, being the other 44 bits (the 28 bits destination field and the 16 bits field used by the third field of a JT65 message) already known by the receiver.

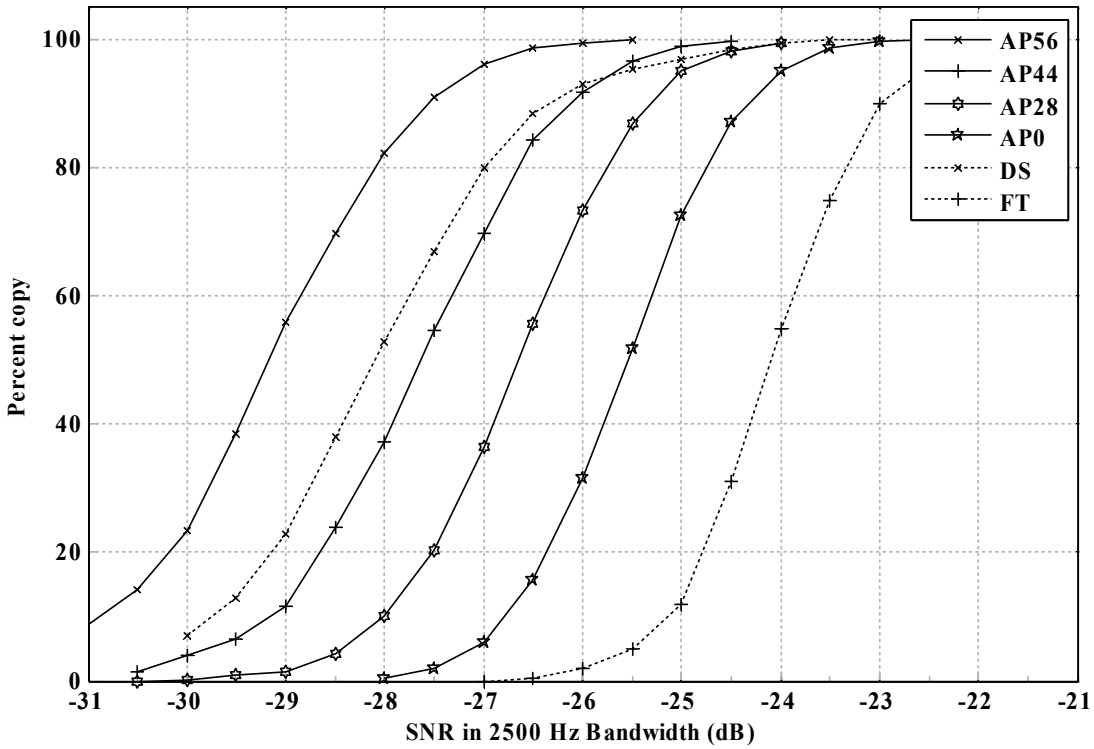

Fig. 5 – Word Error rate of the QRA(12,63) decoder in the Rayleigh channel as a function of the of the a priori (AP) knowledge available to the receiver. The curves AP'nn' indicates the decoder performance when 'nn' bits in a message are known in advance by the receiver. The curve labeled AP0 is the decoder performance when we have no knowledge of the incoming message, as it happens for instance when the message is a plain JT65 text message. This is the same curve that we named QRA in Fig. 4.

The performance of this messaging scheme is shown by the curve labeled AP44 and is just 0.5 dB away from the deep-search algorithm despite we are not using any knowledge other than that we are expecting to receive a call or an answer directed to ourselves. The decoding threshold of a QSO initiated in this way is 3.5 dB better than that of the FT decoder although, to achieve this remarkably good performance, the QRA decoder still uses information received from the communication channel only.

IV – Conclusions

A software package for encoding and decoding the QRA codes I've presented in this paper is going to be published as open source under a GNU GPL license in the hope that it will be useful to the EME digital modes community.

I would like to thank Leif Åsbrink, SM5BSZ for his fruitful suggestions and the encouraging attention he deserved to me during this development. I'd like to thank Joe Taylor, K1JT which along these years always provided me the information and the tools I needed to start this development from.

Finally, a special thank to Andrea Montefusco, IW0HDV which helps me whenever there's some source code to port in OSs other than the one I'm used to work with.

APPENDIX A.

64-ary QRA(12,63) Code Parameters

The irregular QRA(12,63) code on GF(64) whose performance has been shown in Section II and compared with the RS(12,63) code used in JT65, has been designed following the guidelines indicated in Section I and its parameters are listed here below.

Information symbols repetition factors r_k:

```
3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 7
```

Note 1: the first information symbol is repeated three times, the last one 7 times.

Edge permutation matrix:

```
 3,  11,   0,   1,   7,   8,   6,   5,  10,   4,  11,   9,   0,   2,   6,   7,   8,   4,  11,   5,
10,   2,   1,   9,   3,   8,   4,  11,   5,   7,  10,   9,   6,   3,  11,   5,   8,  10,   0,   7,
 9,  11,   4,   2,  10,   6,   8,   1,   9,   7,  11,  10
```

The grouping factor is a=1.
Note 2: the twelve systematic symbols of the code are numbered from 0 to 11. Accordingly to Fig. 1 the first accumulator check input is connected to information symbol #3, the second input to information symbol #11 and so on. The shortest cycle in this code has length 8.

Weights w_k:

```
39,   0,  34,  16,  25,   0,  34,  48,  19,  13, 29,  56,   0,   5,  39,  42,  31,   0,  10,   0,
57,  62,  33,  43,   0,  14,  22,  48,  28,  20,  5,  45,  16,  43,  17,   4,  32,   0,  31,   0,
 0,  28,  57,   0,  18,   0,  60,   0,  10,  31, 57,  27
```

Note 3: weights are given as discrete logarithms in base α, where α is a primitive element of the GF(64) field (see note 5 below for more).

Note 4: both the edge permutation matrix and the weights are sequences of 52 entries while the code has only 51 check variables. The last entry in the tables is not used by the encoder (but it is used by the decoder) and ensures that the implicit 52nd code accumulator check symbol is always 0 whatever the information symbols are.

Note 5: (A very short tutorial on finite fields).
A finite field is a set of elements in which additions and multiplications are defined much as the ordinary operations on a infinite field like that of the integer numbers:
- The addition has a neutral element called 0 such that for any element α in the field $\alpha+0=\alpha$.
- The multiplication has a neutral element called 1 such that for any element α in the field $\alpha\cdot 1=\alpha$.
- Both the addition and the multiplication are symmetric in their operands. $\alpha+\beta=\beta+\alpha$ and $\alpha\cdot\beta=\beta\cdot\alpha$.
- Every element α has an additive inverse which, for finite fields of size $Q=2^q$, is α itself ($\alpha=-\alpha$)
- Every element $\alpha\neq 0$ has a multiplicative inverse α^{-1} such that $\alpha\cdot\alpha^{-1}=1$.
- Taken a field element α, with $\alpha\neq 0$ and $\alpha\neq 1$, we can form the sequence of elements α, $\alpha\cdot\alpha=\alpha^2$, $\alpha\cdot\alpha\cdot\alpha=\alpha^3$, etc.... Each of the elements of this sequence must belong to the field but, since the number of the elements if finite, not all the powers of α are distinct.
We call α a primitive element of a field the element for which all the powers $\alpha^0, \alpha^1, \alpha^2, \alpha^3, ..., \alpha^{Q-2}$

are distinct elements of the field. Every element but the 0 can be expressed as a power of the primitive element. We define the discrete logarithm in base α of an element β, $\log_\alpha(\beta)$, as the integer number j such $\beta = \alpha^j$. The first entry in the code weights table above (39) means that the weight associated with the first accumulator input is α^{39}.

- Any primitive element α of a finite field is a root of an irreducible polynomial G(x) of degree $q=\log_2(Q)$ with binary coefficients a_q, so that $G(x) = x^q + a_{q-1}\, x^{q-1} + a_{q-2}\, x^{q-2} + \ldots + a_1\, x + a_0 = 0$. This generator polynomial allows to express any element of the field as a linear combination of the first q-1 powers of a primitive element α of the field:

$$0 = 0\cdot\alpha^{q-1} + 0\cdot\alpha^{q-2} + \ldots + 0\cdot\alpha^1 + 0\cdot\alpha^0$$
$$1 = 1\cdot\alpha^0$$
$$\alpha = 1\cdot\alpha^1$$
$$\alpha^2 = 1\cdot\alpha^2$$
$$\ldots$$
$$\alpha^q = a_{q-1}\cdot\alpha^{q-1} + a_{q-2}\cdot\alpha^{q-2} + \ldots + a_1\cdot\alpha^1 + a_0\cdot\alpha^0 \text{ (as dictated by the generator polynomial)}$$

$$\alpha^{q+1} = \alpha\cdot\alpha^q = a_{q-1}\cdot\alpha^q + a_{q-2}\cdot\alpha^{q-1} + \ldots + a_1\cdot\alpha^2 + a_0\cdot\alpha^1 =$$
$$a_{q-1}(a_{q-1}\,\alpha^{q-1} + a_{q-2}\,\alpha^{q-2} + \ldots + a_1\,\alpha^1 + a_0\,\alpha^0) + a_{q-2}\,\alpha^{q-1} + \ldots + a_1\,\alpha^1 + a_0\,\alpha^1 =$$
$$(a_{q-1}+a_{q-2})\,\alpha^{q-1} + (a_{q-1}a_{q-2}+a_{q-3})\alpha^{q-2} + \ldots + (a_{q-1}a_1+a_0)\alpha^1 + (a_{q-1}a_0)\alpha^0$$
$$\ldots$$

Thus any element of the field can be expressed by a unique q-uple of binary digits.

For the QRA(12,63) code presented here the primitive element α of the field is a root of the polynomial is $P(x)=x^6 + x + 1$, so that $\alpha^6 + \alpha + 1 = 0$ or $\alpha^6 = \alpha + 1$ and the relationship between the element fields, their discrete logarithm and their binary representation is shown in the following table:

El	Log	Binary	El	Log	Binary	El	Log	Binary	El	Log	Binary
0	-	000000	α^{15}	15	101000	α^{31}	31	100101	α^{47}	47	100111
1	0	000001	α^{16}	16	010011	α^{32}	32	001001	α^{48}	48	001101
α^1	1	000010	α^{17}	17	100110	α^{33}	33	010010	α^{49}	49	011010
α^2	2	000100	α^{18}	18	001111	α^{34}	34	100100	α^{50}	50	110100
α^3	3	001000	α^{19}	19	011110	α^{35}	35	001011	α^{51}	51	101011
α^4	4	010000	α^{20}	20	111100	α^{36}	36	010110	α^{52}	52	010101
α^5	5	100000	α^{21}	21	111011	α^{37}	37	101100	α^{53}	53	101010
α^6	6	000011	α^{22}	22	110101	α^{38}	38	011011	α^{54}	54	010111
α^7	7	000110	α^{23}	23	101001	α^{39}	39	110110	α^{55}	55	101110
α^8	8	001100	α^{24}	24	010001	α^{40}	40	101111	α^{56}	56	011111
α^9	9	011000	α^{25}	25	100010	α^{41}	41	011101	α^{57}	57	111110
α^{10}	10	110000	α^{26}	26	000111	α^{42}	42	111010	α^{58}	58	111111
α^{11}	11	100011	α^{27}	27	001110	α^{43}	43	110111	α^{59}	59	111101
α^{12}	12	000101	α^{28}	28	011100	α^{44}	44	101101	α^{60}	60	111001
α^{13}	13	001010	α^{29}	29	111000	α^{45}	45	011001	α^{61}	61	110001
α^{14}	14	010100	α^{30}	30	110011	α^{46}	46	110010	α^{62}	62	100001

Additions between elements are simply the xor of the binary representations of the arguments. Multiplications can be carrier out noting that each element of the field (but the 0) can be seen as the power of a primitive element and that, when two elements are multiplied, their exponents must be summed (modulo 63) and then the primitive element exponentiated to this sum to get the final result.

For what concerns the code properties, the choice of the generator polynomial is irrelevant, being any particular choice just a different map of the binary q-uples which represent the field elements.

APPENDIX B.

MAPP Decoding of Q-ary LDPC codes. The Message Passing Algorithm.

LDPC codes can be efficiently decoded by the message passing algorithm, a method strictly connected with what in the '90s were already known by the artificial intelligence community under the name of Pearl's "Belief Propagation" algorithm [9].
The MAPP (Maximum A Posteriori Probability) decoding problem we are concerned with is rather simple to state:

- Given the observations on the received signal, any eventual a priori knowledge we have on the transmitted symbols of a codeword and for each information symbol in a codeword:

A. Compute the a posteriori, symbol-wise probability that a symbol was sent,

B. Select as the best estimate of the transmitted symbol the symbol value which looks more likely and maximizes the statistical probability computed at point A.

Although simply to state, when the underlying math of the problem is explored in detail, the task looks extremely complex and, before the message passing algorithm became known, it was indeed something really formidable to solve.
Consider a transmission system that encodes messages in N symbols codewords, that each of the codeword symbols x_n , (n=1..N), is encoded mapping each symbol to a signal waveform and then sent over a memoryless communication channel which add some noise to these waveforms.
At the receiving site, the receiver correlates the received signal with the (known) waveforms used by the transmitter and, for each symbol of the codeword, produces a set of observations s_n (n=1..N) which gives some information on the transmitted symbols.
If we are using a Q-ary code, a MFSK modulation which maps each symbol value to one of the available modulation tones and an incoherent demodulator, each of the s_n observations is in fact a vector of size Q of real numbers and we usually denote it with the underlined symbol $\underline{s}_n$ to indicate that it is a vector.
Under these assumptions and accordingly to the formulation of the MAPP decoding problem we would like to:

A. For every value X_1, X_2, …, X_N the transmitted symbols x_1,x_2,...x_N could assume, compute the a posteriori codeword probability:

Prob {$x_1=X_1$, $x_2=X_2$, $x_3=X_3$, … $x_N=X_N$ given the observations $\underline{s}_1$, $\underline{s}_2$, $\underline{s}_3$, …, $\underline{s}_N$)

B. For every value X_n that the symbol x_n can assume compute the symbol-wise probability:

Prob {$x_n=X_n$ given the observations $\underline{s}_1$, $\underline{s}_2$, $\underline{s}_3$, …, $\underline{s}_N$)

C. For each symbol x_n , choose $\hat{X}_n$ as the the best estimate of the transmitted symbol x_n so that:

$\hat{X}_n$ = value which maximizes Prob {$x_n=X_n$ given the observations $\underline{s}_1$, $\underline{s}_2$, $\underline{s}_3$, …, $\underline{s}_N$)

It is known that, in general, the exact solution of this problems is of exponential complexity.
In step A, in example, we have to compute a function for every possible combination of the transmitted information symbols, a task which is prohibitive when a codeword has more than very few (>3) tens of information bits. Furthermore, if this step looks not sufficiently complex, we then have to compute the probability at step B which can be computed summing the probability at step A over all the combination of symbols but the one we are interested in (this operation is called probability marginalization) and to do this summation for each of the codeword information symbols.

The message passing algorithm [4] [12] is a powerful tool which exploits the fact that the a posteriori probability Prob $\{x_1=X_1, x_2=X_2, x_3=X_3, \dots x_N=X_N$ given the observations $\underline{s}_1, \underline{s}_2, \underline{s}_3, \dots, \underline{s}_N)$, and any of its marginal distribution, can be computed quickly and exactly when the Tanner graph of the code is a tree and it can be factorized into products which can be orderly evaluated.

Accordingly to Bayes' rule about conditional probabilities we know that Prob {x given y} (we will indicate this probability with the symbol p(x|y)) is:

$$p(x|y) = p(y|x)\, p(x) / p(y)$$

We note that if we want to look for the x that maximize p(x|y), the factor p(y) appears in the equation as a multiplicative constant and we can ignore it simply writing:

$$p(x|y) \quad \propto \quad p(y|x)\, p(x)$$

with the meaning that the a posteriori probability p(x|y) of x given the observation y is proportional to the probability p(y|x) of the observation y given x, which is called the likelihood of y given x, multiplied by the a priori probability p(x) of the variable x.
Thus we can rewrite the a posteriori symbols probability of a codeword as:

$$p(x_1, \dots x_N \mid \underline{s}_1,\dots, \underline{s}_N) \quad \propto \quad p(\underline{s}_1, \underline{s}_2, \underline{s}_3, \dots, \underline{s}_N \mid x_1, x_2, \dots x_N)\, p(x_1, x_2, \dots x_N)$$

For a memoryless channel each observation $\underline{s}_j$ depends only on the transmitted symbol x_j and the likelihood $p(\underline{s}_1, \underline{s}_2, \underline{s}_3, \dots, \underline{s}_N \mid x_1, x_2, \dots x_N)$ can be factorized as a product of terms $p(\underline{s}_j \mid x_j)$, j=1..N.
We also note that the a priori probability $p(x_1, x_2, \dots x_N)$ depends both on the code we are using and on the knowledge we may have on any of the transmitted information symbols.
In example if the code we are using imposes the parity check equation $x_1+x_{10}+x_{54} = 0$ we know that this probability $p(x_1, x_2, \dots x_N)$ must be null for any codeword in which the equation $x_1+x_{10}+x_{54} = 0$ is not satisfied. This indicates that also $p(x_1, x_2, \dots x_N)$ can be factorized in a product of terms and we can write:

$$p(x_1, x_2, \dots x_N) = f_1(\underline{X}_1)\, f_2(\underline{X}_2) \dots f_M(\underline{X}_M)\, p_1(x_1) p_2(x_2) \dots p_K(x_K)$$

where: 1) each of the terms $f_m(\underline{X}_m)$ represents a code constraint (a parity check equation) which depends on a subset $\underline{X}_m = \{ x_{m1}, x_{m2} .. x_{mM} \}$, m=1..M, of codeword symbols and evaluate to 1 or 0 if the symbols involved in the set $\underline{X}_m$ satisfy the m-th parity check equation of the code or not, and 2) $p_1(x_1)p_2(x_2)...p_K(x_K)$ represent the a priori probability we have of the information symbols contained in a codeword.
Under the above hypothesis we can rewrite the a posteriori probability of a codeword given the

channel observations, the code constraints and the a priori probabilities of the codeword symbols as a product of terms:

$$p(x_1, x_2, \dots, x_N \mid \underline{s_1}, \underline{s_2}, \dots, \underline{s_N}) = \prod_{n=1..\mathrm{N}} p(\underline{s_n} \mid x_n) \cdot \prod_{m=1..\mathrm{M}} f_m(\underline{X_m}) \cdot \prod_{k=1..\mathrm{K}} p_k(x_k) \quad (1)$$

The marginal, symbol-wise a posteriori probability of the symbol x_k , for any k=1..K, can be then computed summing the codeword probabilities over all the codeword symbols except the symbol x_k:

$$p(x_k \mid \underline{s_1}, \underline{s_2}, \dots, \underline{s_N}) = \sum_{x_1} \sum_{x_2} \dots \sum_{x_{k-1}} \sum_{x_{k+1}} \dots \sum_{x_N} \prod_{n=1..\mathrm{N}} p(\underline{s_n} \mid x_n) \cdot \prod_{m=1..\mathrm{M}} f_m(\underline{X_m}) \cdot \prod_{k=1..\mathrm{K}} p_k(x_k)$$

or more shortly as:

$$p(x_k \mid \underline{s_1}, \underline{s_2}, \dots, \underline{s_N}) = \sum_{\neg x_k} \prod_{n=1..\mathrm{N}} p(\underline{s_n} \mid x_n) \cdot \prod_{m=1..\mathrm{M}} f_m(\underline{X_m}) \cdot \prod_{k=1..\mathrm{K}} p_k(x_k) \quad (2)$$

where with $\sum_{\neg x_k} \dots$ we indicate that the summation is done over all the variables except x_k.

Eq. (1) shows that the APP (A Posteriori Probability) of a codeword is simply a product of factors which involves a set of variables and we can represent it in a graphical way using a Tanner graph as we do with codes: we indicate each variable $x_{1,}...,x_N$ with a circle, each factor of the equation with a box and then we connect the circles (the equation variables) to the boxes (the equation factors) they are referred by with edges. The factors graph we obtain in this way is identical to the Tanner graph of the code with just a small difference: the code variables are now connected to the additional factors $p(\underline{s}_j|x_j)$ which represent the channel observations and the factors $p_k(x_k)$ which represent the a priori information we have on code the information symbols.
The factors graph for the RA code presented in Section I is shown in Fig. 6.
The message passing algorithm exploits the factorization shown in the factors graph to compute iteratively the a posteriori probabilities we are interested into in an very efficient way.
To carry out this task it simply considers variables and check equations as sources and receivers of messages. Such messages are probability distributions. They are computed accordingly to the type of the graph nodes they are involved into and iteratively sent along the graph edges.

Each iteration of the message passing algorithm consists of two steps:

a. compute messages directed from boxes to circles given the messages received by circles,

b. compute messages directed from circles to boxes given the messages received by boxes

The message passing algorithm is started at step a) initializing the messages directed towards each graph circle to the probability distributions coming from the channel observations and any a priori information (the boxes labeled $p_n(x_n)$ and $p(\underline{s}_n|x_n)$). As in this initialization phase no information is still available from the code checks $f_1,...,f_M$, all the boxes $f_1,...,f_M$ are initialized to send to the circles they are connected to a message which corresponds to a uniform probability distribution.
At step b), once every circle in the graph has been provided by messages on any of the edges it is connected to, for any of the edges that connects it to a box, it computes a message directed to that box as the product of all the incoming messages but the message coming from that box.

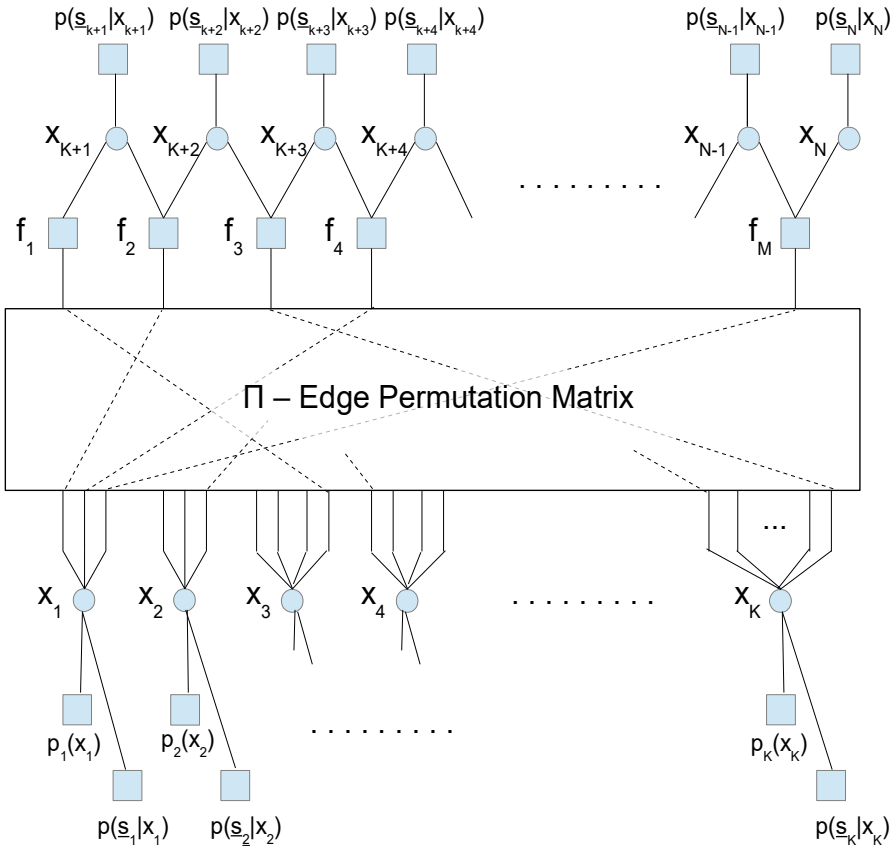

Fig. 6 – The factors graph for the computation of a posteriori probabilities in a RA code. The circles x_n represent the code symbols. The boxes f_m represent the factors associated with the code parity check constraints. The boxes labeled $p_k(x_k)$ indicate the factors associated with the codeword symbols a priori information and the boxes labeled $p(\underline{s}_n|x_n)$ represent the symbols likelihoods (the channel observations).

If we indicate with v_n the generic variable (circle) of the graph, with C(n) the set of the checks it is connected to in the factor graph, with $msg(c_m \rightarrow v_n)$ the message received by the variable v_n by the code check c_m and with $msg(v_n \rightarrow c_m)$ the message sent by v_n to c_m, the variable to check message update rule of step b. can be concisely expressed as:

$$msg(v_n \rightarrow c_m) \propto \prod_{j \in C(n),\, j \neq m} msg(c_j \rightarrow v_n) \quad , \quad n = 1..N \,, \forall\, m \in C(n) \qquad \text{v} \rightarrow \text{c step}$$

This update rule simply states that the probability of a product of (statistically independent) terms is proportional to the product of their probabilities, i.e. that $p(x)=p_1(x)p_2(x)...p_{...}(x)$, therefore, whenever we have some statistically independent evaluations of the variable v_j we can output their product as result. The proportionality constant is determined noting that the output $msg(v_n \rightarrow c_m)$ must be a probability distribution and its elements must sum to unity. Therefore the actual messages output by the v→c step are normalized accordingly to:

$$msg'(v_n = V_k \rightarrow c_m) = \frac{msg(v_n = V_k \rightarrow c_m)}{\sum_{j=1..Q} msg(v_n = V_j \rightarrow c_m)} \quad , k=1..Q$$

where $V_1, V_2, ..V_Q$ are the values a variable can assume and Q is the cardinality of the code alphabet.

After the first iteration the checks to variables messages $msg(c_m \rightarrow v_n)$ of step a. are updated noting that if we need to compute the probability of a variable involved in a parity equation, instead of making a product of probability functions, we have to make convolutions.
Suppose, in example, that we want to compute the probability of the variable x_1 given the constraint $x_1+x_2+x_3=0$ (mod2) and the probabilities $p(x_2)$ and $p(x_3)$. To compute the probability that x_1 assumes a particular value X_1 we have to consider all the cases for which $X_1 + x_2+x_3=0$, that's to say sum the product of the probabilities that $x_2=0$ and $x_3=X_1$, $x_2=1$ and $x3=X_1+1$ (note that we are working with sums mod2) and so on.
If we indicate with V(m) the set of variables connected to the check c_m, the $msg(c_m \rightarrow v_n)$ update rule of step a. is therefore:

$$msg(c_m \rightarrow v_n) = \sum \prod_{j \in V(m),\, j \neq n} msg(v_j \rightarrow c_m) \qquad m = 1..M, \forall n \in V(m) \qquad c \rightarrow v \text{ step}$$

where we have indicated with the symbol $\sum \prod \cdots$ the convolution of its arguments.

After a given number of iterations are made the a posteriori probability of each variable is computed as the product of all the messages coming from the factors it is involved with:

$$app(v_n) \propto \prod_{j \in C(n)} msg(c_j \rightarrow v_n)$$

Note that this computation is identical to the $v \rightarrow c$ step with the only difference that now the product is made on all the inbound edges connected to a variable.
Finally, the last step of the decoding process consist in selecting as the best estimate of the transmitted symbols, the variable value which maximizes the a posteriori probability computed in this way. Once this symbols selection is done we check if all the parity equations of the code are satisfied and declare a successful decode whenever this happens. If at least one parity equation is not satisfied we have two choices: 1) continue with the $v \rightarrow c$ and $c \rightarrow v$ steps iteration to produce new (and possibly more accurate) values for the symbols a posteriori probabilities or, 2) stop the algorithm after a maximum number of iteration is reached and declare a decoding failure.
It should be noted that the message passing algorithm computes the exact marginal a posteriori symbol probabilities only in the case that the factors graph is a tree, that's to say in the case it has no cycles. When the code graph has cycles, as it happens for any practical code, the algorithm do not produce the results an exhaustive computation of all the possible $2^{K \cdot \log_2(Q)}$ cases would deliver. Yet it has a complexity which, unlike exhaustive evaluations, goes linearly with the codeword length and it is still asymptotically optimal.

Despite this linear dependence on the codeword length, the message passing algorithm is quite computational intensive, especially for Q-ary codes in which Q is large.
In the $v \rightarrow c$ step we have to compute M products between probabilities distributions of size Q, for a total of about $M Q \bar{C}$ real numbers multiplications per iteration, where $\bar{C}$ denotes the

average number of factors a variable is connected to.
The c → v step is even more complex as, for any code factor and for any code variable we have to compute a convolution between $\bar{V}$ probability distributions of size Q, where $\bar{V}$ indicates the average number of variables per code factor, for a total of $M\ Q^2\ \bar{V}$ real numbers multiplications per iteration and therefore with a quadratic complexity over the code alphabet size Q. Luckily, these convolutions can be computed in a faster way using an algorithm that is similar to the FFT (Fast Fourier Transform) based convolution where time sequences are transformed in the frequency domain, the frequency domain spectral components are multiplied together and then transformed back in the time domain. The only difference with FFT-based convolutions is that, in the case we need to convolve sequences indexed over symbols belonging to a finite field GF(Q), we use the FWHT (Fast Walsh-Hadamard Transform) in place of the FFT. When computed in this way each convolution requires $3Q\ \log_2(Q)$ operations and a significant computational saving is available when Q>16.

Bibliographic References

[1] http://physics.princeton.edu/pulsar/K1JT/ - Joe Taylor - K1JT
[2] http://www.dj5hg.de/digitalmodes/emepsk_userguide.pdf - Klaus von der Heide – DJ5HG
[3] "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1" – C. Berrou, A. Glavieux, P. Thihimajshima – ICC '93 IEEE Conference on Communications, May 1993.
[4] "Information Theory, Inference and Learning Algorithms" - David J.C. MacKay – Cambridge University Press, 2003
[5] "Irregular Repeat-Accumulate Codes" - H. Jin, A. Khandekar, R. McEliece - 2nd Conference on Turbo Codes – Brest, France, September 2000
[6] "A recursive approach to low complexity codes" - R.Tanner, IEEE Transactions on Information Theory, September 1981
[7] "Design of serially concatenated systems depending on the block length" - M. Tuechler, IEEE Transaction on Communications, Feb. 2004
[8] "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes" - Stephan ten Brink, IEEE Transaction on Communications, October 2001
[9] "Turbo Decoding as an Instance of Pearl's Belief Propagation Algorithm" - R. McEliece, D. MacKay, J.F. Cheng, IEEE Journal on Selected Areas in Communications, February 1998.
[10] "Open Source Soft-Decision Decoder for the JT65 (63,12) Reed-Solomon Code" – Steven Franke, Joe Taylor – QEX May/June 2016 issue, also available here: http://physics.princeton.edu/pulsar/K1JT/FrankeTaylor_QEX_2016.pdf
[11] "Capacity Approaching Codes for Non-Coherent Orthogonal Modulation" – A. Fabregas, A. Grant - IEEE Transactions on Wireless Communications, November 2007
[12] "Factor Graphs and the Sum-Product Algorithm" - F.R. Kschischang, B.J. Frey, H.A. Loeliger, IEEE Transactions on Information Theory, February 2001