Reconstruction of Punctured Convolutional Codes

Mathieu Cluzeau
ENSTA

Matthieu Finiasz
ENSTA

***Abstract*—We present here a new technique to reconstruct punctured convolutional codes from a noisy intercepted bitstream. Compared to existing techniques our algorithm has two major advantages: it can tolerate much higher noise levels in the bitstream and it is able to recover the best possible decoder (in terms of decoding complexity). This is achieved by identifying the exact puncturing pattern that was used and recovering the parent convolutional code.**

***Index Terms*—convolutional codes, puncturing, reconstruction**

Most digital communications are both encoded and encrypted. For this reason, in order to be able to perform a cryptanalysis, it is necessary to decode intercepted data. Usually this data is encoded using a standardized algorithm and it is thus assumed that the attacker can decode as efficiently as the legitimate recipient. However, it can happen that non-standard techniques are used. In this case, the code reconstruction problem needs to be addressed. In this article, we only focus on communications encoded with convolutional codes.

Before the generalization of turbo codes use, concatenated codes were present in most digital communication standards as they offered the best error correction capability. This is for example the case for digital TV (DVB-T) where a Reed-Solomon code is concatenated to a convolutional code. Reverse engineering such codes thus requires to first reconstruct convolutional codes and in particular punctured convolutional codes as these are the most widely used for concatenation.

This article is not the first to deal with reconstruction of convolutional codes. The first to investigate this problem were B. Rice and E. Filiol [7], [8], [12]. More recently J. Barbier also look at this problem [1], [2] and the work by M. Côte and N. Sendrier [5] gives a complete analysis of these techniques. We will give more details about these works in Section II-A and Section III. Concerning puncturing, the technique we present is however the first one to recover the parent code and the puncturing pattern that was used.

This article is composed of three main sections. The first section gives the main definitions and notations about convolutional codes and punctured convolutional codes that are used in the rest of the article. In the second part, we present a new technique for reconstructing a convolutional code from a noisy intercepted bitstream, even for high noise levels. Eventually, we present a technique to recover the original convolutional code hidden behind a punctured convolutional code. This new technique allows to recover the *optimal* decoder for this code.

I. Convolutional Codes and Puncturing

In this section we give the base definitions and notations we use in the rest of this article.

A. Convolutional Codes

An (n,k) convolutional code is defined by a $k \times n$ matrix G of polynomials $G_{i,j}$ in $F_2[D]$. Thus, an (n,k) convolutional code has an information rate of $\frac{k}{n}$. An important parameter of convolutional codes is their *constraint length* which corresponds to the total size of their internal memory. If $d_i = \max_{j\in[0,n-1]} \deg(G_{i,j})$ then the constraint length of the convolutional code is $m = \sum_{j=0}^{k-1} d_i$. This parameter is important as the complexity of the Viterbi decoding algorithmis proportional to 2^{m+k} which means that higher constraint length codes are much more costly to decode.

B. Puncturing

Puncturing a convolutional code consists in transmitting only part of the output of the code, following a regular puncturing pattern. This technique is used to easily build (N,K) codes starting from a simple $(n,1)$ convolutional code. The puncturing pattern is usually presented under the form of a binary matrix of size $n \times K$. Considering K input bits, the orignal code outputs K times n bits which correspond to the K columns of the puncturing matrix: a 1 in the matrix means to transmit the bit, a 0 to remove it. This pattern is repeated for each block of K input bits resulting in an information rate of $\frac{K}{N}$ where N is the Hamming weight (the number of 1s) of the puncturing matrix. This puncturing operation will give an output equivalent to an (N,K) convolutional code.

1) Blocked Code: even without any puncturing, an $(n,1)$ convolutional code can be viewed as an (nK,K) code, for any value of K. The resulting (nK,K) code is called the K times *blocked code* and the original $(n,1)$ code is called the *parent code*. Starting from a code of matrix $G = (G_0, ..., G_{n-1})$ the blocked code has a matrix G' defined by nK polynomials $P_{i,j}$ such that: $G_i = \sum_{j=0}^{K-1} D^j P_{i,j}(D^K)$. Each of the n polynomials defining G is split in K different polynomials corresponding to terms of degree $j \mod K$ of G_i. Then, the matrix G' is defined by:

$$\begin{cases} G'_{i,j} = P_{j \bmod n, \lfloor \frac{j}{n} \rfloor - i} & \text{if } n \times i \leq j \\ G'_{i,j} = D \times P_{j \bmod n, \lfloor \frac{j}{n} \rfloor - i + K} & \text{if } n \times i > j \end{cases}$$

[1] This work was supported by the French DGA in the context of the AINTERCOM contract.

$$\left[\begin{array}{cc|cc|cc} P_{0,0} & P_{1,0} & P_{0,1} & P_{1,1} & P_{0,2} & P_{1,2} \\ \hline DP_{0,2} & DP_{1,2} & P_{0,0} & P_{1,0} & P_{0,1} & P_{1,1} \\ \hline DP_{0,1} & DP_{1,1} & DP_{0,2} & DP_{1,2} & P_{0,0} & P_{1,0} \end{array}\right]$$

Figure 1. Matrix of a $(2,1)$ code blocked 3 times to obtain an equivalent $(6,3)$ convolutional code. If $G_0 = 1 + D + D^3 + D^4$ and $G_1 = 1 + D^2 + D^3 + D^5$ we obtain $P_{0,0} = 1+D$, $P_{0,1} = 1+D$, $P_{0,2} = 0$, $P_{1,0} = 1+D$, $P_{1,1} = 0$, and $P_{1,2} = 1 + D$.

$$\left[\begin{array}{cc|c|c|c|c} P_{0,0} & P_{1,0} & & P_{1,1} & P_{0,2} & \\ \hline DP_{0,2} & DP_{1,2} & & P_{1,0} & P_{0,1} & \\ \hline DP_{0,1} & DP_{1,1} & & DP_{1,2} & P_{0,0} & \end{array}\right]$$

Figure 2. Matrix of a $(2,1)$ code punctured using a the puncturing matrix $\left(\begin{smallmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{smallmatrix}\right)$ resulting in a $(4,3)$ equivalent convolutional code.

Matrix G' will have a form similar to that of Figure 1. Puncturing then simply consists in selecting N columns out of the nK columns of the blocked matrix G' and removing the other ones so as to obtain an (N,K) code (see Figure 2).

Note that it is also possible to puncture generic (n,k) codes in a similar manner but we will not consider this case in this article and only focus on punctured $(n,1)$ codes which are the most widely used.

II. Reconstruction of Convolutional Codes

Reconstructing a convolutional code can have different meanings. A complete reconstruction consists in recovering the matrix G that was used for encoding, but this is not always possible. In practice, one can recover the vector space $\mathcal{G}$ generated by G over $F_2(D)$, but then, any polynomial basis of $\mathcal{G}$ is a possible encoder. However knowing any basis of $\mathcal{G}$ is enough to correct errors, so we first recover the space $\mathcal{G}$ and then try to deduce the correct matrix G.

A. Existing Technique

In 1995, B. Rice was the first to deal with the problem of reconstructing a convolutional code and proposed an algorithm [12] to solve this problem for $(n,1)$ convolutional codes when there are no errors. Later, E. Filiol also investigated this problem and designed algorithms [7], [9] to solve this problem for any convolutional code. Moreover, these algorithms could be adapted to handle noisy bitstreams. All of these algorithms are based on the Berlekamp-Massey algorithm [3], [11] and consist in finding a long enough sequence of noise-free output bits. Recently, Barbier et al. have further improved these techniques [1], [2] by using algorithms based on randomized Gaussian elimination, making it possible to deal with noise more efficiently.

These algorithms perform very well in the absence of noise: they have a much better complexity than the algorithm we propose here and are much more straightforward. However, when introducing noise in the intercepted bitstream, looking for a noise-free sequence can become very difficult, if not impractical. In this case our technique will outperform the previous ones as it will still run in a few seconds on any standard computer, and this, as long as the noise level remains reasonable.

Using completely different techniques based on the Expectation Maximization algorithm, J. Dingel and J. Hagenauer [6] were also able to reconstruct convolutional codes in the presence of noise. However, for the moment, their technique can only be applied to some convolutional codes.

B. Using Valembois' Algorithm

Valembois' algorithm [13] is based on the Canteaut-Chabaud information set decoding algorithm [4] and makes it possible to efficiently find words in the dual of a linear code from a set of noisy code words.

In the case of convolutional codes a similar notion of dual exists: any polynomial vector of degree d orthogonal to the generator matrix G of a convolutional code can be converted to a binary "dualword" of length $n \times (d+1)$. This dualword is orthogonal to any shift by a multiple of n bits of the output bitstream of the code. This means that if the intercepted noisy bitstream is split in blocks of length $n \times (d+1)$ and each block is seen as a noisy code word, Valembois' algorithm should be able to recover the binary dualword and thus the orthogonal polynomial vector.

1) The Reconstruction Algorithm: the technique we use is quite simple: we know that for block lengths which are sufficiently large multiples of n, Valembois' algorithm should recover some dualwords. We can thus test if a given length ν is a sufficiently large multiple of n or not by splitting the intercepted bitstream in blocks of ν bits and applying Valembois' algorithm.

Our algorithm tests all values of ν incrementally, starting from 1: if dualwords are found, then ν is probably a multiple of n. Once a first value ν_0 giving dualwords has been found we continue incrementing ν until a second value ν_1 also giving dualwords is found. The value of n is most probably $\nu_1 - \nu_0$. This can be further checked by running the algorithm for other multiples of $\nu_1 - \nu_0$.

Once n has been found and checked, it is possible to go on with the reconstruction. We just need some dualwords and we already should have plenty: any dualword found for ν_0, ν_1 or any other verification length can be used. For each of these dualwords, a polynomial vector orthogonal to G can be derived. Let us denote by $\mathcal{H}$ the vector space generated by these polynomial vectors.

If enough dualwords have been found, the space $\mathcal{H}$ should be of dimension $n-k$ so that $\mathcal{G} = \mathcal{H}^\perp$. In practice, Valembois' algorithm outputs dozens of dualwords in a second: during our tests, even with the highest noise levels and the largest codes, we were always able to generate the whole orthogonal space $\mathcal{G}^\perp$. This means that we can already determine the value of k and that $\mathcal{G}$ has been found. We simply select a polynomial basis G' of $\mathcal{G}$ and can use it to correct errors.

2) Canonical Reduction of the Reconstructed Matrix: any polynomial basis of the space $\mathcal{G}$ gives a matrix equivalent to G, that is, a matrix producing the same code and enabling the same error correction. However, depending on the matrix, decoding can be more costly than with the original matrix G. Our method can output any matrix G', but we want the best possible matrix, the matrix with the smallest possible constraint length, as it will give the fastest decoder.

Luckily for us, algorithms computing such a matrix already exist. For any space $\mathcal{G}$ such as ours, a *canonical basis* is a basis with, among other properties, the smallest possible *external degree*. The external degree of a basis is the sum of the degrees of the k elements of this basis and thus exactly correspond to the constraint length of the code. We thus simply need to build a canonical basis G'' from the basis G' we computed. This is done in two steps:

- first make the basis G' *basic* by computing its Smith form (*cf.* [10] p. 39),
- then *reduce* this basic basis to decrease the degrees of the elements while keeping the basis basic (*cf.* [10] p. 58).

Once these two steps have been performed the basis G'' we obtain is canonical. This gives us the best possible decoder for the convolutional code. However, we still cannot be sure that $G'' = G$: many canonical bases of a same space $\mathcal{G}$ can exist. The only thing we know is that G'' can correct noise at least as efficiently as G.

C. Practical Experiments

The technique we proposed makes it possible to recover an equivalent matrix G'' for any (n, k) convolutional code of matrix G. The only input required is a noisy output bitstream from the code. We ran a variety of experiments considering a binary symmetric channel with different crossover probabilities. Table I contains some examples of the noise levels our algorithm can handle.

We compared our results to those obtained using the method of J. Barbier, G. Sicot and S. Houcke [2]. In particular, we used the same $(4, 3)$ code of constraint length 8 as in [1] and were able to reconstruct it with a crossover probability of 0.02. Their algorithm was successful only up to 0.01. We could not find any figures for their algorithm with crossover probability higher than 0.02, so comparison for parameters where our algorithm can handle noises of 0.1 was impossible.

The results obtained by J. Dingel and J. Hagenauer in [6] through a completely different method are quite hard to compare to ours. They mostly focus on the number of iteration their algorithm requires to converge and thus correctly reconstruct the code. The example they give is for a $(4, 1)$ code with constraint length 5 and a crossover probability of 0.1. For the same code we were able to reconstruct it up to a crossover probability of 0.15. We thus believe that our algorithm has performances similar to their algorithm. However, it appears that, depending on the code, their algorithm does not always perform so well. This is not the case for our algorithm which, as we verified experimentally, behaves very homogeneously for codes of identical rates and constraint length.

Table I
HIGHEST ERROR RATES τ FOR WHICH OUR ALGORITHM COULD RECONSTRUCT AN EQUIVALENT CODE MATRIX G'' IN A FEW SECONDS.

(n, k)	constraint length	τ
$(2, 1)$	2	0.15
$(2, 1)$	6	0.1
$(3, 1)$	6	0.12
$(4, 1)$	3	0.2
$(4, 1)$	5	0.15
$(4, 1)$	6	0.1
$(4, 3)$	8	0.02
$(5, 2)$	14	0.03

III. RECONSTRUCTION OF PUNCTURED CONVOLUTIONAL CODES

Some previous works on convolutional code reconstruction [1], [8] already deal with the problem of punctured codes. What they state is that reconstructing a punctured convolutional code simply requires to reconstruct the equivalent (N, K) code (see Section I-B for notations) and use this code for noise correction. This is perfectly true and the algorithm we presented in previous section can serve the same purpose. However, decoding with the (N, K) code is not efficient. For standard punctured codes, the (N, K) code has the same constraint length m as the parent $(n, 1)$ code[2]. However, the cost of the Viterbi decoding algorithm does not only depend on m: for an (n, k) code, the cost is proportional to 2^{m+k}. This means that decoding with the (N, K) code instead of the parent $(n, 1)$ code costs 2^{K-1} times more. This additional cost is usually not a problem as small values of K are generally used, however, being able to recover the $(n, 1)$ code would have a real added value. Unfortunately, recovering this parent code is not an easy task. Even if the puncturing pattern is known, no efficient algorithm exists to recover the $(n, 1)$ parent code. The algorithm we present now is the first to achieve this in polynomial time.

A. Description of the Problem

In order to recover the parent $(n, 1)$ code from the reconstructed (N, K) code it is necessary to recover a matrix of the blocked (nK, K) code with the exact same structure as depicted in Figure 1. This requires two things:

- find the puncturing pattern,
- find a matrix with the suitable structure (matching polynomials at the correct positions).

Concerning the puncturing pattern we could not find any technique to guess it efficiently. Almost all patterns correspond to a possible $(n, 1)$ parent code but most of them are of large constraint length. We therefore propose to test all the possible patterns and, as we will see later, select the pattern giving the best results. First we have to guess n and then tests all the $\binom{nK}{N}$ possible puncturing patterns. Most of the times n will be small, so we start with 2 and test all puncturing patterns,

[2] Puncturing can sometimes decrease the constraint length of a code, but such a puncturing will necessarily affect the correction capabilities of the code. Therefore, all standard punctured codes preserve their constraint length.

$$Z = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ D & 0 & 0 \end{pmatrix}, \; M = \begin{pmatrix} P_{0,2} & P_{1,2} \\ P_{0,1} & P_{1,1} \\ P_{0,0} & P_{1,0} \end{pmatrix}$$

$$\text{and} \quad G = \left(\; Z^2 \times M \; \middle| \; Z \times M \; \middle| \; M \; \right)$$

Figure 3. Mathematical structure of a blocked code matrix G. Here, $K = 3$.

then test $n = 3$ and all corresponding patterns and so on until a satisfying solution is found.

For each pattern we test, we now need to find all the possible matrices (with the correct blocked structure) corresponding to the matrix we reconstructed. Luckily for us, the structure corresponding to a blocked code matrix can be put in equations.

Property 1: Let Z be the $K \times K$ matrix consisting of an upper diagonal of 1, a D in the bottom left corner and 0 elsewhere (see Figure 3), then, for any blocked code matrix G on $F_2[D]$, there exists a $K \times n$ matrix M on $F_2[D]$ such that G is the block matrix $(Z^{K-1} \times M | Z^{K-2} \times M | \cdots | Z \times M | M)$.

Property 2: Let G_i denote the i-th column of matrix G, then any matrix verifying $\forall i \in [0, N-n-1], G_i = Z \times G_{i+n}$ has the structure of a blocked matrix.

Thanks to Property 2, we know how to detect a blocked matrix. We now need to be able to enumerate all possible matrices of a given code, that is, all polynomial bases of a vectorial subspace $\mathcal{G} \subset F_2(D)^N$ of dimension K. Starting from a basis G of $\mathcal{G}$, any other basis G' can be obtained as $G' = P \times G$ for a given $K \times K$ transition matrix P on the field of rational fractions $F_2(D)$. Our algorithm will thus start from the reconstructed matrix and find all possible transitions matrices P on $F_2(D)$ that could transform it into a matrix with the structure of a blocked code.

B. Building an Equation System and Solving it

A given puncturing pattern is associated to a function $\phi : [0, N-1] \rightarrow [0, nK-1]$ which maps the index i of a column of the punctured matrix to its position $\phi(i)$ in the original matrix. Starting from a reconstructed $K \times N$ matrix G'' it can be expanded to a matrix G' such that $G'_{\phi(i)} = G''_i$ and G'_j is unknown for any $j \notin \{\phi(i), i \in [0, N-1]\}$.

What we are looking for is the set of all possible transition matrices P on $F_2(D)$ such that $P \times G'$ has the structure of a blocked code matrix. However, G' has some unknown columns: this means that these columns (which have been punctured) do not add any constraints on P, the only constraints on P come from columns of G' which have the same index modulo n. We get the set of equations:

$$\forall i, j \text{ such that } \phi(i) = \phi(j) \bmod n,$$
$$Z^{(\phi(j)-\phi(i))/n} \times P \times G''_{\phi(j)} = P \times G''_{\phi(i)} \quad (1)$$

These equations are the only constraints on P: once all equations of the form (1) are satisfied for a matrix P, $P \times G''$ expands in a matrix $P \times G'$ which has the form of a blocked code matrix (minus the punctured columns).

1) Solving the System: we write down all equations derived from (1) in a large system and now need to solve it. In fact, each equation of the form (1) is an equality between two vectors of size K and thus yields K equations linking coefficients of P: these equations are linear equations over the field $F_2(D)$. Also, if three indexes $\phi(i)$, $\phi(j)$ and $\phi(l)$ are equal modulo n, three equations of the form (1) can be written, but only two of them are independent. In general, if ℓ indexes are equal modulo n, $\ell - 1$ independent equations can be written, yielding $(\ell - 1)K$ equations in our linear system.

In order to write the whole system we thus split the N indexes $\phi(i)$ in n classes modulo n and write all independent equations for each of these classes. We end up with a linear system in K^2 unknowns (the coefficients of P) with a number of equations that will depend on the puncturing pattern.

This linear system can easily be solved and the solution space $\mathcal{S}$ is a vector space on the field $F_2(D)$: each element of this space is a valid transition matrix P. Thus, for any $P \in \mathcal{S}$ the matrix $P \times G''$ can be expanded in $P \times G'$ which has the correct blocked structure.

2) Recovering the Parent Code: for a given transition matrix P on $F_2[D]$, from the n last columns of the expanded matrix $P \times G'$ we can deduce the n polynomials of the equivalent $(n, 1)$ convolutional code: some of these columns might have been punctured but can easily be deduced from other columns of $P \times G'$ by multiplication by a power of Z^{-1}. Let us denote by Ψ the function which transforms a transition matrix P into the $n \times 1$ matrix of the deduced $(n, 1)$ convolutional code. For any $P \in \mathcal{S}$, $\Psi(P)$ gives a decoder for the convolutional code with the selected puncturing ϕ. Once again, we want to find the fastest possible decoder for a given puncturing pattern, that is, the matrix P yielding the smallest degree $\Psi(P)$.

Theorem 1: The image $\Psi(\mathcal{S})$ of $\mathcal{S}$ by Ψ is a vector space over the field $F_2(D)$ (even though Ψ is not really an homomorphism) and any polynomial basis of $\mathcal{S}$ is transformed by Ψ into a generating set of $\Psi(\mathcal{S})$.

Proof: The solution space $\mathcal{S}$ and the function Ψ have the following, easy to verify, properties:

- $\forall q \in F_2(D)$, if $P \in \mathcal{S}$ then $qP \in \mathcal{S}$,
- $\forall q \in F_2(D), \Psi(qP) = q^K \Psi(P)$,
- if $P \in \mathcal{S}$ and $P' \in \mathcal{S}$ then $P + P' \in \mathcal{S}$,
- $\Psi(P + P') = \Psi(P) + \Psi(P')$,
- for any integer i (positive or negative), if $P \in \mathcal{S}$ then $Z^i \times P \in \mathcal{S}$ (this can be seen in equation (1)),
- $\Psi(Z^i \times P) = D^i \, \Psi(P)$,
- Ψ is invertible.

These properties are enough to prove Theorem 1. ■

Theorem 1 tells us that, after solving our system, we simply need to select a polynomial basis $\mathcal{B} = \{P_0, ..., P_b\}$ of $\mathcal{S}$. Then for each element P_i compute its image $\Psi(P_i)$ and select the polynomial vector of smallest degree in the vector space generated by the $\Psi(P_i)$ vectors. This last step can be done by using our basis reduction algorithms from Section II-B: if we input the generating set $\{\Psi(P_i)\}$ and put it in canonical form it will output a canonical basis of $\Psi(\mathcal{S})$, which is exactly what we are looking for. We simply select the lowest degree vector of this basis (which will usually contain a single vector): this is the best possible decoder for the puncturing ϕ we are testing.

In order to recover the effective puncturing pattern that was used on our intercepted bitstream we simply test all possible puncturing patterns and for each of them build a system, solve it to obtain a basis $\mathcal{B}$ and transform it with Ψ, then use a canonical reduction algorithm to get the best decoder for this pattern. We then simply need to select among all tested puncturing patterns the one giving the most efficient decoder, that is, the lowest degree $(n, 1)$ parent code.

C. Complexity Analysis and Practical Results

The algorithm we propose works in three steps:

- build a linear system in K^2 unknowns and find a basis of its solution space with a Gaussian elimination,
- transform each element of this basis using function Ψ,
- put the obtained matrix in canonical form.

The linear system in K^2 unknowns contains between $K \times (N - n)$ and $K \times (N - 1)$ equations (depending on the puncturing pattern). As N is always of the same order as K the Gaussian elimination will require $O(K^6)$ polynomial operations. However, the degree of the polynomials can increase with K and thus make the complexity even higher. This is not the case in practice: we compute GCDs during the elimination to keep degrees low and this seems to be sufficient to maintain a complexity in $O(K^6)$ for this step. We were able to verify this quite accurately during our experimentations.

The complexity of the second step depends on the dimension of the solution space $\mathcal{S}$. This dimension is always a multiple of K as for any solution P, all matrices of the form $Z^i \times P$ are also solutions. In practice, this dimension is between K and $(n - 1) \times K$ for all puncturing patterns (for a $(2, 1)$ parent code it is always K). Function Ψ computes n polynomials from an nK matrix it multiplies by P and has a complexity of $O(nK^2)$. The complexity of this step is thus $O(n^2K^3)$ operations on polynomials of small degree. Note that if some puncturings were incompatible with a given (N, K) code this dimension would then be 0. This was never the case during our experimentations, confirming the fact that for a given code, all puncturing patterns yield a compatible parent code.

The final step consists in computing the canonical form of an $(n - 1)K \times n$ matrix. This could cost $O(n^3K^2)$ but costs much less in practice as the canonical form is of size $1 \times n$.

The dominating complexity is the first step in $O(K^6)$ and this is also the most costly step in practice. It takes from 50% to 90% of the whole running time depending on the parameters. Testing one puncturing pattern takes a fraction of a second but many patterns have to be tested. Our algorithm runs in a few minutes for a $(2, 1)$ parent code punctured into a $(9, 8)$ code, which corresponds to $\binom{16}{9} = 11\,440$ patterns to test, but takes only a few milliseconds for the same $(2, 1)$ parent code punctured into a $(4, 3)$ code.

Concerning the uniqueness of the solution, usually one puncturing pattern clearly stands out and offers a much shorter constraint length than other puncturing patterns, but it can happen that two (maybe more) patterns give identical results (parent codes with the same constraint length). In such a case it is impossible to decide which pattern was used in practice which leaves an ambiguity on the decoded sequence. This however does not affect the error correction capability of the code as both solutions give equivalent decoders.

Our algorithm can probably be adapted to punctured (n, k) parent codes, but the final canonical reduction certainly has to be modified. We however did not look further into this problem as we could not find any real life example of punctured code not coming from an $(n, 1)$ parent code.

IV. Conclusion

We presented two new algorithms for the reconstruction of convolutional codes from an intercepted noisy bitstream. The first one applies to generic convolutional codes and enables reconstruction in the presence of high noise levels with a relatively short bitstream (a few thousand bits are enough). This algorithm runs in a fraction of a second. The second algorithm applies to punctured convolutional codes and is the first algorithm to recover the $(n, 1)$ parent code thus giving the best possible decoder. The running time of this algorithm spans from a few milliseconds to a few minutes (or more in some extreme cases) depending on the puncturing pattern.

References

[1] J. Barbier. *Analyse de canaux de communication dans un contexte non coopératif.* Phd thesis, École Polytechnique, November 2007.

[2] J. Barbier, G. Sicot, and S. Houcke. Algebric approach of the reconstruction of linear and convolutional error correcting codes. In *CCIS 2006*, 2006.

[3] E.R. Berlekamp. *Algebraic coding theory.* McGraw-Hill, 1968.

[4] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: application to primitive narrow-sense BCH codes of length 511. *IEEE Transaction on Information Theory*, 44(1):367–378, January 1998.

[5] M. Côte and N. Sendrier. Reconstruction of convolutional codes from noisy observation. In *IEEE Conference, ISIT 09*, 2009. To appear.

[6] J. Dingel and J. Hagenauer. Parameter estimation of a convolutional encoder from noisy observations. In *IEEE Conference, ISIT 07*, pages 1776–1780, 2007.

[7] É. Filiol. Reconstruction of convolutionnal encoders over GF(q). In Michael Darnell, editor, *Cryptography and Coding ; proceedings of the 6th IMA conference*, number 1355 in LNCS, pages 101–109. Springer-Verlag, 1997.

[8] É. Filiol. Reconstruction of punctured convolutional encoders. In *International Symposium on Information Theory and its Applications (ISITA'00)*, 2000.

[9] É. Filiol. *Technique de reconstruction en cryptologie et théorie des codes.* Phd thesis, École Polytechnique, March 2001.

[10] R. Johannesson and K. Sh. Zigangirov. *Fundamentals of convolutionnal coding.* IEEE Press, 1999.

[11] J.L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[12] B. Rice. Determining the parameters of a rate $\frac{1}{n}$ convolutional encoder over $GF(q)$. In *Third International Conference on Finite Fields and Applications*, Glasgow, 1995.

[13] A. Valembois. *Détection Décodage et Reconnaissance des Codes Linéaires Binaires.* Phd thesis, Université de Limoges, October 2000.