TSG-RAN Working Group 1 meeting No. 5
June 1-4, Cheju, Korea

TSGR1-641/99

Agenda Item: **Adhoc 4 item 7 Channel Interleaving**

Source: **Siemens**

Title: **Properties of optimised puncturing scheme**

Document for: **Information and Discussion**

Introduction

For puncturing after first interleaving an optimised puncturing algorithm has been proposed in [1]. This algorithm is proposed to be used together with the FS-MIL for the first interleaver and the modified FS-MIL interleaver as proposed in [2] for the second interleaver. The puncturing algorithm [1] is repeated here, together with an optimisation which gives the same results but will be less complex for a possible implementation.

The performance of this puncturing scheme is investigated in comparison to an alternative proposal [3] i.e. the algebraic interleaver. We then compare the complexity and flexibility of the two proposals.

As a conclusion we will see that the optimised Puncturing scheme has the most advantageous properties.

Principle of optimised algorithm

The goal of a good puncturing algorithm is to spread punctured bits as evenly as possible. This was the driving principle for the algorithm in [2] as well. This can best be obtained by puncturing every n^{th} bit (for non integer puncturing rates sometimes every n^{th} and sometimes every $n+1^{st}$ bit). We suggest to apply this principle also for puncturing after interleaving, but there is one constraint: We have to distribute punctured bits on all frames evenly. For example, assume 80 ms interleaving and a puncturing rate of 1:6. By puncturing every 6^{th} bit we would only puncture column 0,2,4,6 but not 1,3,5,7 which is of course impossible. To balance puncturing between columns, we have to change the puncturing interval sometimes (here once) to avoid hitting always the same columns. This is shown in Fig. 1. Bold horizontal arrows show puncturing distance of 6 and the thick hollow arrow shows puncturing distance 5 to avoid hitting the first column twice. After having punctured every column once, the pattern can be shifted by 6 rows to determine the next bits to be punctured (vertical arrows). Obviously this is equivalent to puncture every 6^{th} bit in each column and shifting puncturing patterns in different columns relative to each other. The actual puncturing is done using the well known puncturing algorithm from S1.12, and the shift can be realised by loading the initial offset in the variable e with a proper value.

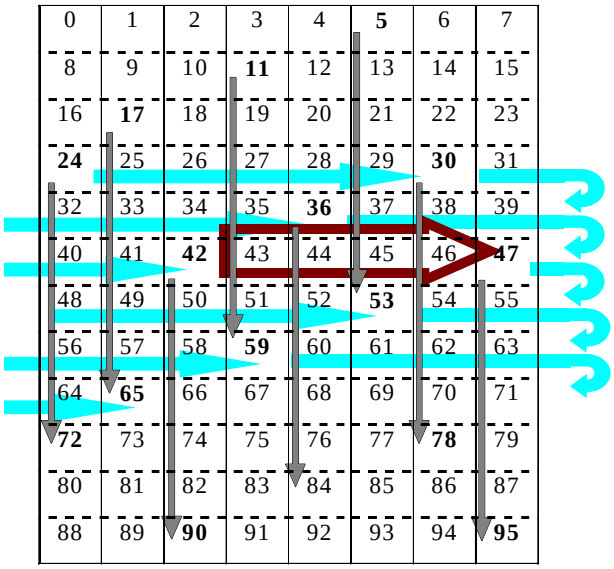

Figure 1: Principle of optimised puncturing

Formulas for optimised algorithm

We now repeat the formulas for the optimised algorithm: Denote the number of bits in one frame before rate matching by N_c, the number of bits after rate matching by N_i, the frame number by k, and the number of interleaved frames by K ($0 \le k < K$). We mainly consider the case when $N_c > N_i$, that means puncturing, but the formulas will be applicable for repetition as well. Shifting could then be achieved with the following formula:

-- calculate average puncturing distance
q:= *ëN_c/(ôN_i-N_cô)û -- where ë û means round downwards and ô ô means absolute value.*

if q is even -- *avoid hitting the same column twice:*

then q' = q –gcd(q, K)/K -- where gcd (q, K) means greatest common divisor of q and K

-- note gcd can be easily computed using bit manipulations, because K is a power of 2.

-- note that q' is now not an integer, but calculations with q' can be easily done using binary fixed point

-- arithmetic (or integer arithmetic and a few shift operations) because q' is a multiple of 1/8

else

q' = q

endif

–calculate S , S represents the shift of the row

for *i = 0* to *K-1*

$S(R_K(éi*q'ù \bmod K)) = (éi*q'ù \text{ div } K)$ -- *where é ù means round upwards.* $R_K(k)$ *reverts the interleaver as in [7]*

end for

Then, e_{offset} can be calculated as

$e_{offset}(k) = ((2*S)* y + Nc) \bmod 2Nc$

e_{offset} (k) is then used to preload *e* in the rate matching formula in [4] (there the constant value $e = N_C$ is used).

Note that the values of S(k) are not calculated in their natural order. In an implementation, optionally, instead of using the preceding formula the value of i belonging to a specific combination of q and k can be pre-stored in a table. Note that the table also includes the effect of re-mapping the column randomising achieved by $R_K(k)$. Then S can then be calculated from i as follows:

$S(k) = éi*q'ù \text{ div } K$

i	K	1	2		4				8							
	k	0	0	1	0	1	2	3	0	1	2	3	4	5	6	7
(q mod K)	0	0	0	1	0	2	3	1	0	4	6	2	7	3	5	1
	1		0	1	0	2	1	3	0	4	2	6	1	5	3	7
	2				0	1	3	2	0	2	1	3	5	7	6	4
	3				0	2	3	1	0	4	6	2	3	7	1	5
	4								0	1	5	4	7	6	3	2
	5								0	4	2	6	5	1	7	3
	6								0	2	3	1	7	5	6	4
	7								0	4	6	2	7	3	5	1

Table 1: Value of i to be used to calculate S depending on k, q and K. Note that the table only contains 3 bit values and is still somewhat redundant

This algorithm will obtain the perfect puncturing as if puncturing using the rate matching algorithm was applied directly before interleaving, if the puncturing rate is an odd fraction i.e. 1:5 or 1:9. For other cases, adjacent bits will never be punctured, but sometimes the distance between punctured bits may have to be reduced by one in order to avoid hitting only even frames. Note that this algorithm should be applied to bit repetition as well. While repeating adjacent bits is not as bad as puncturing them, it is still advantageous to distribute repeated bits as evenly as possible.

The basic intention of these formulas is to try to achieve equidistant spacing of the punctured bits in the original order, but taking into account the constraint, that the bits have to be punctured equally in different frames. This may make it necessary to reduce the puncturing distance by 1 sometimes. The presented algorithm is optimum in the sense, that it will never reduce the distance by more than 1, and will reduce it only as often as necessary. This gives the best possible puncturing pattern under the above mentioned constraints.

Performance considerations

A decisive property of a puncturing scheme is that punctured bits are distributed as evenly as possible (in the original ordering of the bits). Otherwise some bits will be coded weaker and therefore suffer a higher BER and will thus spoil the performance. An extreme case would be puncturing of adjacent bits, which should clearly be avoided. The optimised puncturing algorithm was designed with exactly these points in mind and will achieve a next to optimum performance, i.e. the puncturing is (almost) as evenly distributed as is the case if puncturing was applied before of first interleaving.

In the following table we present the minimum distance between consecutive punctured bits. The first row shows the puncturing rate, the second one the average distance (inverse of puncturing rate), the third row the actually achieved minimum puncturing distance for the case of 10 ms interleaving (which is the same as for the case that puncturing is performed before first interleaving) and the last row the minimum puncturing distance if interleaving is done over more than 10ms. Note that almost always the optimum puncturing distance is achieved, in particular for the puncturing rates above 17% (indicated by bold figures). Of course for high puncturing rates, where the performance is most noticeable affected by the puncturing pattern, it is most important to have an even puncturing.

Puncturing rate in %	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
average distance	100	50	33,3	25	20	16,7	14,3	12,5	11,1	10	9,09	8,33	7,69	7,14	6,67	6,25	5,88	5,56	5,26	5
Min. dist. 10 ms interl.	100	50	33	25	20	16	14	12	11	10	9	8	7	7	6	6	**5**	**5**	**5**	**5**
Min. dist. > 10 ms interl.	99	49	33	25	19	15	13	11	11	9	9	7	7	7	5	5	**5**	**5**	**5**	**5**

These minimum puncturing distances were agreed to be made available for the different schemes, unfortunately these data are lacking for the algebraic interleaving and puncturing scheme. In order to be able to compare the schemes we have derived the puncturing distances from the interleaver and puncturing files which have been made available by the proponents and calculated the distribution of the distance between successive punctured bits.

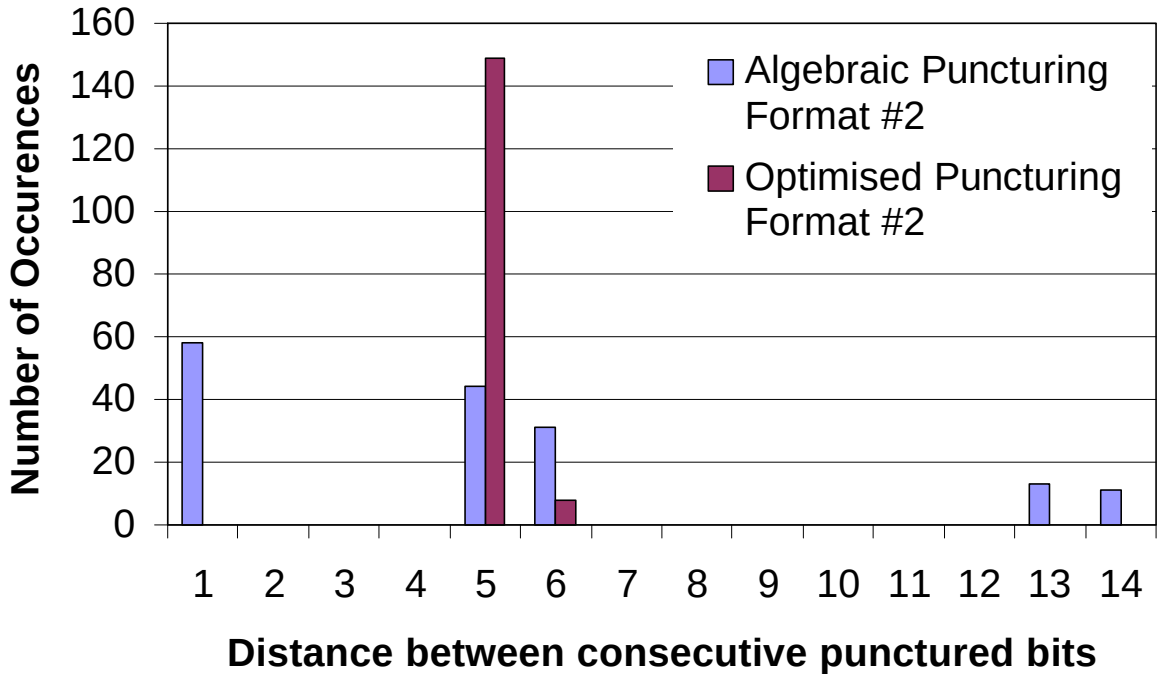

Figure 2: Distribution of punctured bits

In a first approach we have investigated the distribution of punctured bits for simulation format 2 and a puncturing rate of 19.8 %. Figure 2 shows the results, specifically the distance between consecutive punctured bits considering the input data stream of the interleaver. We achieve a balanced distribution of punctured bits and a very low variance of distance between consecutive punctured bits using the optimised puncturing scheme. Note that the *average* puncturing distance is always 1/p where p is the puncturing rate, because if two punctures are moved closer together then they have to be moved further apart from the next puncture. As can be seen easily there is no puncturing of consecutive bits using the optimised puncturing algorithm. But the occurrences of punctured adjacent bits for the algebraic scheme is very high compared to the occurrences of ‘better’ distances e.g. 5 and 6. Note that it will be the cases of close punctured bits which will spoil the performance, a loss that can not be recovered. Therefore the maximum puncturing distance does not tell anything about the performance, as bit errors can not be compensated by other extra secure transmitted bits. We think

this results clearly show the better balanced performance of the optimised puncturing scheme. This also explains the better performance in AWGN simulations compared to the algebraic scheme.

Complexity estimation

The proposed puncturing algorithm is very similar to the second interleaver or the bit merging process which is proposed for the algebraic interleaver. In both cases, the current puncturing algorithm was copied and slightly modified. Therefor the amount of work to be done per bit i.e. the effort to be spent in the inner loop is almost equal. The initial overhead to initialise the loops may be different, but this is performed rarely, at most every 10 ms and as explained above, by using small tables it can be implemented very efficiently, so it will be a negligible contribution to the overall complexity.

Considering the complexity of the associated interleavers, it can be observed, that both the Algebraic and the modified FS-MIL interleaver are built of two stages, the column randomising and the row randomising step, but the actual formulas for the column and row randomising differ and both steps are executed in the first interleaver for the algebraic interleaver while row randomising (here called intra frame interleaving) is done in the second interleaver for the MIL. However, the modified MIL allows a very efficient implementation in particular (but not exclusively) if a DSP solution is considered. This is due to the fact, that the MIL can be implemented using a few short lookup tables and otherwise modulo 2^x addressing. Both operations are typically supported by modern DSPs and can also be implemented efficiently in hardware. The Algebraic interleaver however requires more sophisticated modulo calculations using prime numbers which are harder to implement on both DSPs and dedicated hardware, because both inherently use binary representations of numbers.

Note that a flexible DSP implementation is very appealing for the following reason: A DSP could either perform a video compression for medium data rates, then the video handy would be used as in standalone operation. Alternatively the same DSP could handle higher data rates which would then be delivered to a PC, i.e. the DSP would not perform any source coding but only deliver the user data. The same argument is valid for a low end terminal that either supports low data rates for speech (running the AMR and additional acoustic operations like filtering, distortion compensation and echo cancellation) or medium data rates when connected to a PC or similar device. With dedicated hardware one could not reuse the resource for these different applications (source coding or extra channel coding and interleaving). Note that video compression algorithms and advanced acoustic algorithms use both a sizeable amount of RAM, which could be used as interleaving RAM as well and also require more and more processing power.

Flexibility, in particular for turbo coding

Two proposals have been presented in Adhoc 5 regarding puncturing for turbo coding [5],[6]. Both make use of the existing puncturing algorithm and introduce small modifications to it to address the specific requirements for turbo coding. As the original puncturing algorithm is also used for the optimised puncturing, both the turbo coding specific modifications can be easily incorporated in a straight forward manner. This will allow to enjoy the corresponding performance gain for punctured turbo codes also in the uplink.

Conclusion

This paper has shown that optimised puncturing has very appealing properties regarding performance, implementation complexity and flexibility as compared to the algebraic puncturing scheme and should therefore be considered for puncturing in the uplink.

References

[1] R1-99203 Optimised Rate Matching after interleaving Siemens

[2] NTT DoCoMo "Modified Rate Matching Algorithm in uplink" 3GPP RAN TSG WG1 Ad Hoc 4; March 15th, 1999

[3] R1-99466 On the Algebraic Channel Interleaver Design Nortel Networks

[4] 25.212 v1.0.0; 3GPP TSG RAN WG1; Multiplexing and channel coding (FDD)

[5] R1-99338 Puncturing Algorithm for Turbo Code LGIC

[6] R1-99388 Optimised puncturing scheme for Turbo coding Fujitsu