THE MODIFIED SOFT INPUT PARITY CHECK TRANSFORMATION ALGORITHM FOR REED SOLOMON CODES

Y. Genga , D.J.J. Versfeld*

* *School of Electrical and Information Engineering,University of the Witwatersrand, Private Bag 3, Wits 2050, Johannesburg, South Africa. E-mail: yuval.genga@students.wits.ac.za, and jaco.versfeld@wits.ac.za.*

Abstract: In this paper, we propose a modification to the recently developed Parity check Transformation Algorithm (PTA) used in the decoding of Reed Solomon codes. This extension of the PTA is referred to as the Modified Parity check Transformation Algorithm (MPTA). The MPTA is developed with the aim of reducing the number of iterations run by the algorithm during the decoding process, and also to improve on the SER performance of the algorithm. Three version of the MPTA are developed in this paper to achieve this goal.

Key words: Reed Solomon codes, Soft decision decoding, Iterative decoding.

1. INTRODUCTION

Reed-Solomon (*RS*) codes [1] are a class of linear block codes that are widely used for applications that range from telecommunications to storage devices. These codes are largely popular because they meet the Singleton bound. What this means is that a hard decision decoder (HDD) based on minimum distance decoding can correct up to $[n-k]/2$ symbols for a given (n,k) *RS* code. Hard decision decoders for *RS* codes include the Berlekamp-Massey (*B-M*) algorithm [2] [3], the Euclidean algorithm [4], and the Berlekamp-Welch algorithm [5]. HDD algorithms for *RS* codes have been shown to be efficient, however, they are significantly outperformed by soft decision decoding techniques.

Koetter and Vardy (*KV*) [6] presented a symbol level decoding algorithm that utilizes the soft reliability information from the channel to make a multiplicity matrix that is fed into the Guruswami Sudan (*GS*) [7] algorithm. The *KV* algorithm outperforms HDD algorithms significantly especially for low rate codes. However, complexity can become prohibitively large when trying to achieve large coding gains with the *KV* algorithm [8].

The Parity check Transformation Algorithm (PTA) is a symbol wise soft decision decoding algorithm that was recently proposed in [9]. The algorithm transforms the parity check matrix, H, of an *RS* code after every iteration depending on the reliability of the symbols. The reliability of the symbols are attained from soft information received at the channel output. In the same paper, the PTA algorithm was shown to outperform the widely used Koetter and Vardy (*KV*) algorithm and the Berlekamp-Massey (BM) algorithm for simulations run in the AWGN channel with the symbols being mapped onto a 16QAM modulation scheme. The performance was measured in terms of the symbol error rate (SER). The major drawback of the PTA is the high number of iterations needed to find the decoded codeword, especially for low SNR cases.

In this paper, modifications to the PTA are introduced with the aim of reducing the number of iterations required to decode. These modifications are used to develop an extension of the PTA, which is referred to as the Modified PTA (MPTA)

The rest of the paper is structured as follows. A basic description of how the PTA decoding algorithm works is given in Section 2. The analysis and modifications to the PTA is explained Section 3. A performance comparison between the PTA and the MPTA algorithms is done in Section 4. Finally, a conclusion is given in Section 5.

2. THE PARITY CHECK TRANSFORMATION ALGORITHM

We now summarize the PTA algorithm [9] to establish notation. Consider an (n,k) *RS* code with a parity check matrix H in the field GF(2^p). Let c be a codeword of length n. The symbols of the codeword c are mapped onto signals based on a selected modulation scheme and then transmitted through an AWGN channel. At the output of the channel a received vector r is obtained. Based on the vector r a reliability matrix R can be constructed. The matrix R is then fed into the PTA decoder.

The steps involved in one iteration of the PTA algorithm are as follows

1. *Getting the symbol reliability*: Find the highest values in each column of the R matrix. From these values, let the k highest values represent the most reliable symbols. The indices of these symbols are denoted by $\mathcal{K}$. The remaining $n-k$ values represent the least reliable symbols and their indices are denoted by $\mathcal{U}$.
2. *The H matrix transformation step*: Transform the H matrix into matrix H^t by performing row operations similar to Gaussian such that H^t is in a rearranged systematic form based on the indices $\mathcal{U}$ and $\mathcal{K}$. The transformation is done in such a way that the indices of $\mathcal{U}$ match the partitioned identity matrix and the $\mathcal{K}$ indices match the parity partition as shown in [9]. A

computationally efficient way of doing this is shown in [10].

3. *Performing the syndrome checks*: For $1 \leq i \leq (n-k)$, the i^{th} row of the matrix H^t can be denoted as H_i^t. A syndrome check is now performed using all the $n-k$ rows of the matrix H by performing hard decision detection on the R matrix to obtain a vector $\hat{r}$, and getting the dot product $\hat{r} \cdot H_i^t$. Due to the rearranged systematic structure of H_i^t, only one symbol with the index in $\mathcal{U}$ participates in the syndrome check. All the k symbols in $\mathcal{K}$ participate in the dot product.

4. *The correction step*: If the dot product is not zero, we decrease the symbol indexed in $\mathcal{U}$ by the value of a correction factor, δ, and decrease the symbols indexed in $\mathcal{K}$ by $\delta/2$. If the syndrome check is satisfied we add the δ to the symbol indexed in $\mathcal{U}$ and $\delta/2$ to the symbols indexed in $\mathcal{K}$.

5. *The update step*: These corrected reliabilities are then used to update the R matrix based on the $\mathcal{U}$ and $\mathcal{K}$ indices. The updated R matrix is then used in the next iteration.

6. *The stopping criterion*: steps 1 to 5 are repeated until all the $(n-k)$ syndrome checks are satisfied, or until a negative value appears in the matrix R.

From [9] it was shown that a smaller correction factor (δ) improves the performance of the algorithm. This, however, comes at the cost of increased number of iterations as the algorithm has to perform more corrections to the received codeword.

3. MODIFICATIONS TO THE PTA ALGORITHM

We now analyze the performance of the PTA and the effect of the modifications to the algorithm. Simulations are based on a (15,7) Reed-Solomon code using BPSK modulation in an AWGN channel. The value of δ used is 0.001.

3.1 Performance of the PTA based on the number of iterations

To better understand how the PTA algorithm works, the early stopping condition used in [9] is not considered. The only stopping criterion for the algorithm is when all syndrome checks are satisfied. This version of the PTA defined by the syndrome check as the only stopping condition is denoted as PTA$^{\gamma}$.
Table. 1 shows the results for simulations run for the PTA$^{\gamma}$ with 1000 codewords at an SNR of 0 dB. The number of codewords with error and the number of iterations required to satisfy the checks are recorded.

Fig. 1 shows simulations for the Hamming distance between the actual codeword and the decoded codeword at the output of each iteration of the PTA$^{\gamma}$. Results in Fig. 1 are representative for the 4 types of codewords decoded using the PTA$^{\gamma}$ in Table 1. The 4 types of codewords represented include: the codewords correctly decoded within 1000 iterations, the codewords decoded correctly with more than 1000 iterations, the codewords decoded incorrectly within 1000 iterations and the codewords decoded incorrectly for over 1000 iterations.

Table 1: Number of errors compared to the number of iterations for 1000 codewords

Number of iterations	detected codeword	Number of codewords
Less than 1000	Correct	853
	Wrong	54
More or equal to 1000	Correct	3
	Wrong	90

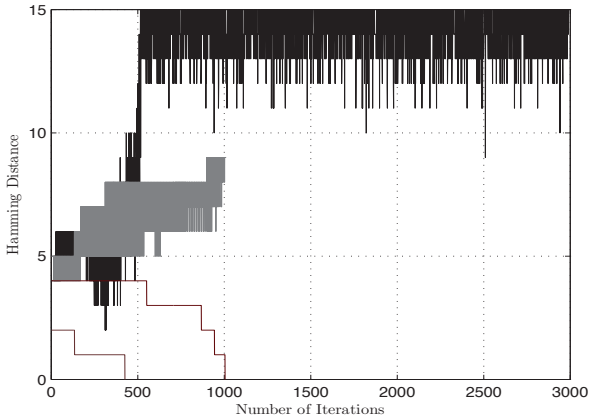


Figure 1: Comparing the Hamming distance to the Number of Iterations.

From Table 1 and Fig. 1 it can be seen that the PTA$^{\gamma}$ algorithm outputs a codeword, regardless if it is correct or incorrect, when it satisfies the syndrome checks. What this means is that more iterations for the PTA$^{\gamma}$ algorithm don't necessarily mean an improved performance with the algorithm. This can be seen especially when the algorithm runs for more than 1000 iterations. From Table 1 it can be seen that most of the syndrome checks for the correct codewords are also satisfied whenever the PTA$^{\gamma}$ runs less than 1000 iterations. Most incorrect codewords are received from the PTA$^{\gamma}$ whenever the algorithm runs for more than 1000 iterations.
From Table 1 only 3 codewords that run for 1000 or more iterations with the PTA$^{\gamma}$ are decoded correctly. This indicates the presence of a threshold in the PTA$^{\gamma}$ which occurs when the algorithm runs for more than 1000 iterations. Fig. 2 investigates the effect of different thresholds on the PTA$^{\gamma}$ by setting different maximum number of iterations (I_{max}) the algorithm can run.

From Fig. 2 it can be seen that the threshold of the PTA$^{\gamma}$ is at I_{max} = 1000 and that it serves as a point of saturation for the algorithm, as the performance appears to remain the same beyond this point. Thus, a set maximum value

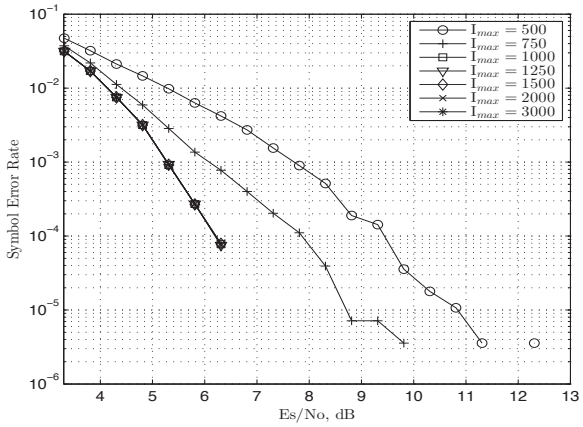


Figure 2: Performance comparison of different values of I_{max}

of 1000 iterations for the PTA$^{\gamma}$ should be used to prevent redundancy.

3.2 *Performance of the PTA based on the number of negative values in R*

A possible reason for the increased errors with more iterations could be the appearance of negative values formed in the reliability matrix, originating from the corrections done during each iteration. Simulations to see how the number of negative values in the reliability matrix corresponded to the actual codeword matching the received codeword at the output of the PTA$^{\gamma}$. The test was run for 1000 codewords. The results of these simulations were recorded in Table. 2.

From Table. 2, it can be seen that for most cases, the

Table 2: Number of errors compared to the number of negative values for 1000 codewords

Reliability Matrix	detected codeword	Number of codewords
Negative found	Correct	7
	Wrong	133
Remains Positive	Correct	858
	Wrong	2

presence of a negative value in the reliability matrix directly affect the number of errors of the received codeword at the output of PTA$^{\gamma}$. This result validates the early stopping condition of the PTA algorithm used in [9]. To prevent errors caused by the negative values, absolute values of the corrected reliabilites are used after each iteration of the PTA$^{\gamma}$. This modification of the PTA$^{\gamma}$ with absolute values is referred to as the absolute PTA$^{\gamma}$ and denoted as aPTA$^{\gamma}$.

3.3 *Performance of the PTA based on the Stagnant check.*

If the absolute values are to be used, the early stopping condition defined in [9] is not applicable as the reliability matrix does not run into a negative value. To prevent the aPTA$^{\gamma}$ from running upto I_{max} iterations in the event it is unable to find the correct codeword, a new early stopping condition is required. A modification of the stagnant check [11], is added to the algorithm to perform this function. The stagnant check acts as a predictive algorithm, by trying to determine if the algorithm will have a decoding failure or a decoding success.
The stagnant check algorithm works as follows

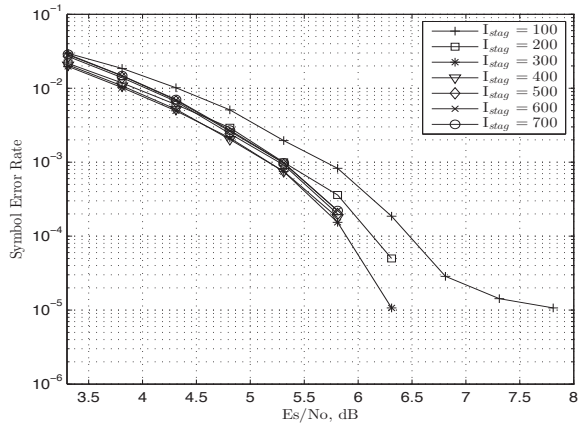
1. At a given iteration I_{α}, the weight of the unsatisfied syndrome checks ($w(s)$) is noted.
2. The value of $w(s)$ is noted for each iteration after I_{α} until a specified iteration I_{stag} is reached.
3. At iteration I_{stag}, a stagnant check is done.
    - If $w(s)$ is the same from I_{α} to I_{stag} then the algorithm is perceived to be stagnant (unable to find the correct codeword), and the decoding process is stopped.
    - If $w(s)$ has changed from I_{α} to I_{stag} then the aPTA$^{\gamma}$ runs until all the syndrome checks are satisfied, or until I_{max} is reached.

The main aim of the stagnant check is to help reduce the number of iterations required by the aPTA$^{\gamma}$. To get an optimum performance of the aPTA$^{\gamma}$ with the stagnant check, simulations are done for different values of I_{stag} and run for $I_{max} = 1000$. Simulations are run for $I_{\alpha} = 1$ and the stagnant check was done at $I_{stag} = 100$, $I_{stag} = 200$, $I_{stag} = 300$, $I_{stag} = 400$, $I_{stag} = 500$, $I_{stag} = 600$, $I_{stag} = 700$. The average number of iterations run by each was noted. The results for these simulations can be seen in Fig. 3.

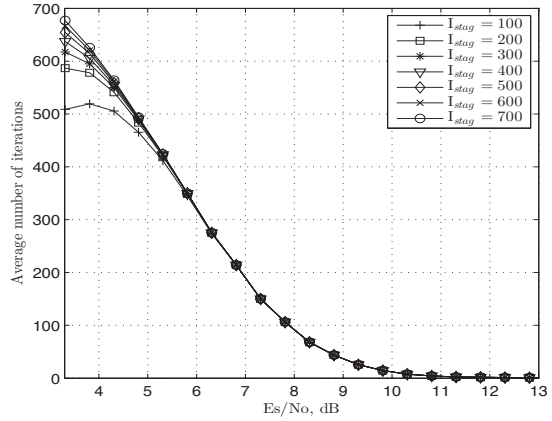
From Fig. 3 it can be seen that the aPTA$^{\gamma}$ with the value of $I_{stag} = 400$ gives the best performance when compared to the rest. The aPTA$^{\gamma}$ with the stagnant check is referred to as the Modified Parity check Transformation Algorithm (MPTA).

3.4 *Performance of the PTA based on the value of the correction factor, δ*

From Fig. 3(b), it can be seen that even with the stagnant check the algorithm still runs for many iterations in low SNR regions. The main reason for this is the use of a small value of δ. The smaller the value of δ the better the perfomance, but this is at the cost of an increased number of iterations as shown in [9]. This is because there is not a large difference in the corrections made by δ in each iteration for the wrong symbols as it attempts to satisfy the syndrome checks. Since the difference in the corrections made by a δ of 0.001 are quite small several iterations would be required to satisfy the checks. Simulations for different values of δ with the aPTA$^{\gamma}$ are done to test for the performance and the average number of iteration required by the algorithm. No stagnant check is used because an accurate representation of the average number of iterations

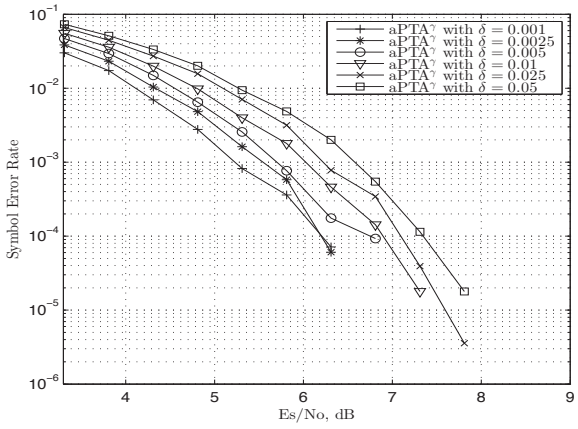


(a) SER Perfomance for different values of I_{stag} for the MPTA

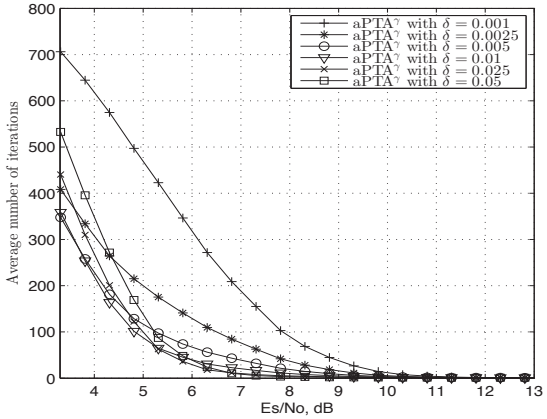


(b) Average number of iterations for the MPTA

Figure 3: Perfomance of the MPTA based on the value of I_{stag}



(a) SER Performance of different aPTA$^\gamma$ with different values of δ



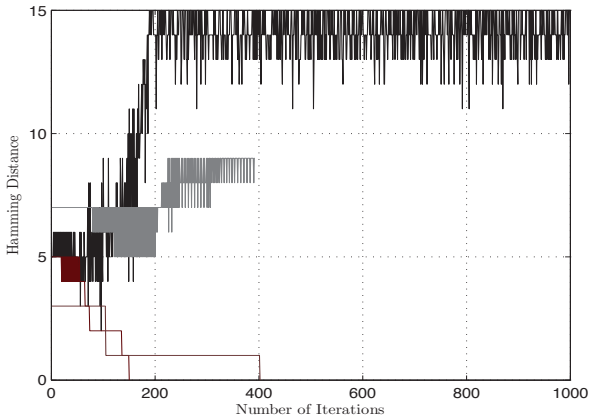
(b) Average number of iterations for the aPTA$^\gamma$ with different values of δ

Figure 4: Perfomance of the aPTA$^\gamma$ based on different values of δ

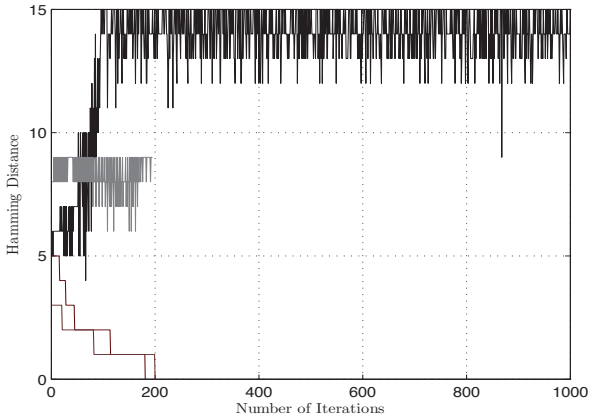
run by the algorithm for each SNR is required. The results for these simulations can be seen in Fig. 4.

Fig. 4 validates the results shown in [9] with a BSPK modulation scheme for the PTA. It can also be seen that the number of iterations run by the aPTA$^\gamma$ between $\delta = 0.005$ and $\delta = 0.001$, increases as the value of δ reduces for lower SNR regions. However, the opposite seems to happen between $\delta = 0.01$ and $\delta = 0.05$. This is due to these values of δ making a large difference in the corrections made to the soft information. This causes the algorithm to change the reliability of the symbols faster, thus passing the correct symbols as it satisfies the checks. It is evident from Fig. 4 that the value of the correction factor should be $\delta < 0.01$ for better SER results to be obtained with the algorithm.
To reduce the number of iterations run by the MPTA, a discreet variable δ approach is added to the algorithm. For this implementation, two scenarios are considered. A case of increasing values of δ, and a case of reducing values of δ. In both cases, the value of δ changes with respect to the number of iterations.
For this analysis, three values of δ are used. The values of δ chosen are $\delta = 0.005$, $\delta = 0.0025$ and $\delta = 0.001$. The reason these values of δ have been selected is because the performance difference between $\delta = 0.001$ and $\delta = 0.005$, from Fig. 4, is about 0.5 dB.
To help with the implementation of this form of discreet variable δs with the algorithm, a similar simulation to the one done in Fig. 1 is done for the PTA$^\gamma$ with $\delta = 0.005$ and $\delta = 0.0025$ for 1000 codewords. This is done to help with the assignment of the number of iterations that each δ receives. The results for these simulations can be seen in Fig. 5, Table 3 and Table 4.
The simulations are let to run for as many iterations until the syndrome checks are satisfied. For Fig. 5, to better see the results, only the first 1000 iterations are shown. Once again, results for only 4 codewords that are representative of most of the other codewords decoded with the PTA$^\gamma$ are used. The 4 types of codewords are highlighhted in the results recorded in Table 3 and Table 4.

From Fig. 5 and Table 4 it can be seen that for $\delta = 0.005$ most of the correct received codewords are found within 200 iterations. Beyond this, the algorithm begins to saturate and is unable to find the correct codeword to satisfy the syndrome checks. For $\delta = 0.0025$ the algorithm



(a) Comparing the Hamming distance to the Number of iterations for $\delta = 0.0025$



(b) Comparing the Hamming distance to the Number of iterations for $\delta = 0.005$

Figure 5: Performance of the PTA$^{\gamma}$ with $\delta = 0.0025$ and $\delta = 0.005$ based on the number of iterations

Table 3: Number of errors compared to the number of iterations for 1000 codewords with $\delta = 0.0025$

Number of iterations	detected codeword	Number of codewords
Less than 400	Correct	808
	Wrong	36
More or equal to 400	Correct	14
	Wrong	142

Table 4: Number of errors compared to the number of iterations for 1000 codewords with $\delta = 0.005$

Number of iterations	detected codeword	Number of codewords
Less than 200	Correct	733
	Wrong	22
More or equal to 200	Correct	24
	Wrong	221

appears to saturate after about 400 iterations as seen in Fig. 5 and Table 3. Therefore, of the 1000 maximum iterations for the MPTA, the number of iterations each δ runs is assigned with respect to the SER perfomance shown in Fig. 4(a) and the performance based on the number of iterations required to satisfy the checks shown in Fig. 1 and Fig. 5. For the case of the increasing δ, the number of iterations is split as follows

- $\delta = 0.001$ runs for the first 625 iterations
- $\delta = 0.0025$ runs the next 300 iterations
- $\delta = 0.005$ runs the last 75 iterations

For the case of the decreasing δ, the number of iterations is split as follows

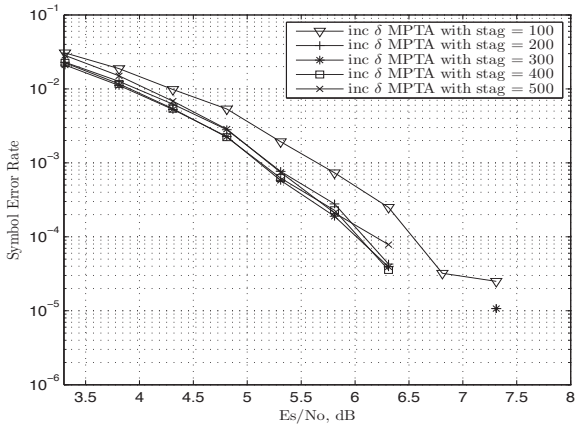
- $\delta = 0.005$ runs for the first 75 iterations
- $\delta = 0.0025$ runs the next 300 iterations
- $\delta = 0.001$ runs the last 625 iterations

Simulations for the discreet variable δ version of the MPTA are done to test for the optimum performance with different values of I_{stag}. These simulations are for SER performance and the average number of iterations required by the algorithm. The values of I_{stag} used in the simulations for the decreasing δ are smaller than those used for increasing δ. The reason for this that for the decreasing δ case, the first value of δ used is 0.005. An earlier stagnant check for $\delta = 0.005$ is required as the corrections it makes to the codeword during each iteration are larger than those of $\delta = 0.001$. As a result it runs into a stagnant state much earlier as shown in Fig. 5. The results of these simulations for the both cases of increasing and decreasing δ can be seen in Fig. 6 and Fig. 7 respectively.

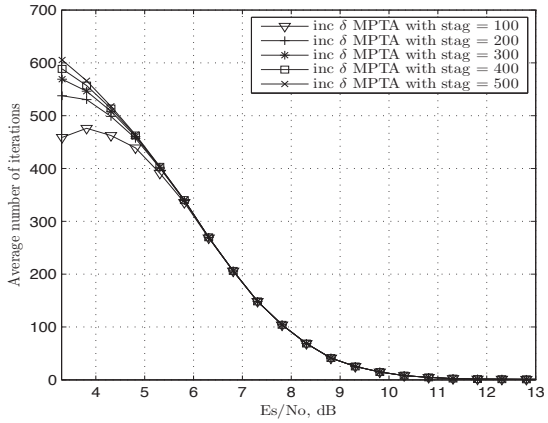
From Fig. 6 it can be seen that the best performance, in terms of SER, is obtained when $I_{stag} = 400$. It is naturally outperformed by $I_{stag} = 100$, $I_{stag} = 200$ and $I_{stag} = 300$ in terms of the number of iterations required for the lower SNR values. Based on the SER performance, the value of $I_{stag} = 400$ was selected for the increasing δ version of the MPTA in the comparative study.
A similar observation can be done in Fig. 7 with $I_{stag} = 80$, as it the gives best performance in terms of SER. In terms of the number of iterations, it is outperformed by $I_{stag} = 20$, $I_{stag} = 40$ and $I_{stag} = 60$ for low SNR values. Based on the SER performance, the value of $I_{stag} = 80$ was selected for the decreasing δ version of the MPTA in the comparative study.

4. PERFORMANCE ANALYSIS

Fig. 8 shows simulations for the performance comparison of PTA to that of the MPTA with a single δ of 0.001 and $I_{stag} = 400$, an increasing δ with $I_{stag} = 400$ and a decreasing δ with $I_{stag} = 80$.
From Fig. 8 it can be seen that the single δ MPTA and the increasing δ MPTA have an almost identical performance and they both outperform the PTA in terms of SER. In



(a) SER Perfomance for Increasing δ MPTA with different values of I_{stag}



(b) Average number of iterations for the increasing δ MPTA for different values of I_{stag}

Figure 6: Performance of the MPTA

terms of the average number of iterations for low SNR areas, the decreasing δ MPTA significantly outperforms all versions of the PTA. However, this is at a cost of about 0.5 dB. This SER performance difference justifies the use of the decreasing δ MPTA when a trade-off between SER performance and number of iterations is required. The increasing δ MPTA runs for less iterations than the PTA and gives a better performance. The single δ MPTA gives a better perfomance than the PTA but gives no difference in the average number of iterations.

5. CONCLUSION

In this paper, we are able to develop three different versions of the MPTA, each with its own trade-offs. The single δ MPTA increases the performance of the algorithm but at no reduction in the number of iterations. The increasing δ MPTA outperforms the PTA in terms of both the SER and the number of iterations required to decode. It also gives an almost identical performance to that of the single δ MPTA while running for less iterations in the low SNR regions. Finally, the decreasing δ MPTA significantly reduces the average number of iterations run by the algorithm by well over 400 iterations. This reduction in the number of iterations comes at the cost of a loss of less than 0.5 dB when compared to the PTA.

ACKNOWLEDGEMENT

The financial assistance of the National Research Foundation (NRF) of South Africa towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF. The financial support of the Centre for Telecommunication Access and Services (CeTAS), the University of the Witwatersrand, Johannesburg, South Africa is also acknowledged.

REFERENCES

[1] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc.Ind. Appl. Maths.*, 1960.

[2] J. Massey, "Shift-register synthesis and BCH decoding," *Information Theory, IEEE Transactions*, 1969.

[3] E. R. Berlekamp, "Algebraic Coding Theory," *New York: McGraw-Hill*, 1968.

[4] S. H. Y. Sugiyama, M. Kasahara and T. Namekawa, "A method for solving key equation for decoding goppa codes," *Information and Control*, 1975.

[5] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," *Patent US 4 633 470*, 1986.

[6] R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," *Information Theory, IEEE Transactions on*, 2003.

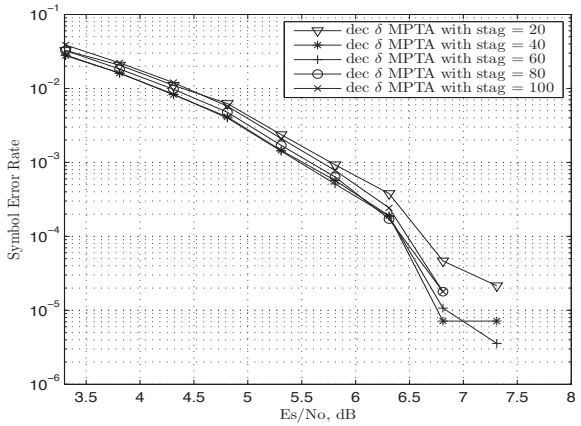
[7] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and Algebraic-Geometry Codes," *Information Theory, IEEE Transactions on*, 1999.

[8] J. Jiang and K. Narayanan, "Iterative Soft-Input Soft-Output Decoding of Reed Solomon Codes by Adapting the Parity-Check Matrix," *Information Theory, IEEE Transactions on*, 2006.

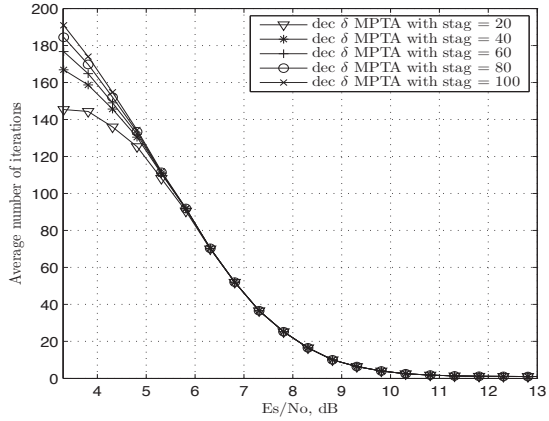
[9] O. Ogundile, Y. Genga, and D. Versfeld, "Symbol level iterative soft decision decoder for Reed-Solomon codes based on parity-check equations," *Electronics Letters*, vol. 51, no. 17, pp. 1332–1333, Aug. 2015.

[10] D. Versfeld, J. Ridley, H. Ferreira, and A. Helberg, "On Systematic Generator Matrices Reed-Solomon codes," *Information Theory, IEEE Transactions on*, vol. 56, no. 6, pp. 2549–2550, June 2010.

[11] B. Shin, S. H. Kim, J. S. No, and D. J. Shin, "New stopping criteria for decoding LDPC codes in H-ARQ systems," in *Information Theory and Its Applications, 2008. ISITA 2008. International Symposium on*, Dec 2008, pp. 1–5.

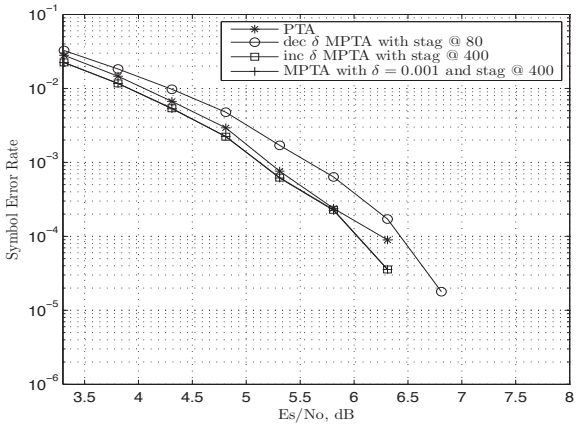


(a) SER Perfomance for decreasing δ MPTA with different values of I_{stag}

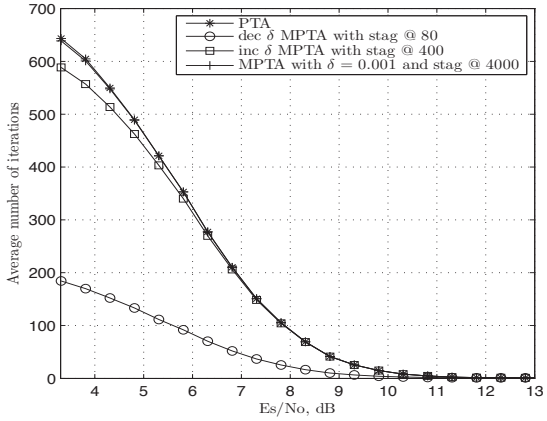


(b) Average number of iterations for the decreasing δ MPTA for different values of I_{stag}

Figure 7: Performance of the MPTA



(a) SER Perfomance for the MPTA and the PTA



(b) Average number of iterations for the MPTA and the PTA

Figure 8: Performance of the MPTA compared to the PTA