4
Experimental Results

In this chapter we present the motivation for our tests and comment on the results obtained through simulation. We start by comparing the performances of different degree distributions, based on the simulated transmission using codes constructed through the standard PEG algorithm applied to degree distributions obtained by [JKM2000]. Then we use the most suitable distribution to compare different construction methods. The reversed variation of the Look-Ahead enhanced version of the PEG algorithm, presented in Chapter 3, is compared to the other methods based on its simulated performance.

4.1
Motivation

The experimental work of this dissertation consists of testing the performance of IRA codes constructed with different methods, all seeking to avoid forming cycles in the graph. All codes are generated following the degree distributions provided in the paper by Khandekar, Hui Jin and McEliece [JKM2000] where IRA codes were first proposed.

We conjecture that the bit error performance of a code is related to the number of length-four cycles in its Tanner graph and that the occurrence of undetected errors is related to the code's minimum distance. We first observe some properties of different graphs by verifying through direct analysis the number of length-4 cycles and estimating the minimum distance through encoding of low-weight messages.

We compare five different methods that stem from the PEG algorithm [HEA2005] in the sense that the edges are traced in order of growing left-degree, with each variable node entering the graph only after the previous node has no more empty sockets. Also, all methods attempt to prevent length-four cycles by expanding trees with the current variable node as the root before placing each edge, and then joining the root to one of the most distant [1] check-nodes.

The five construction methods are:

Standard PEG (PEG-ST) is exactly what is described in algorithm 1.

[1] *distant* as in Definition 8

PEG Level-one (PEG-L1) is the simplest method: the edges are traced with the restriction of not creating length-four cycles and preventing neighbors from being joined by more than one edge.

Non-Greedy PEG (PEG-NG) differs from **Level-one** by tracing edges only to the check-nodes with the lowest degree in the current graph setting.

Look-Ahead PEG (PEG-LA) is an enhanced version of the Standard PEG that anticipates the effect that an additional edge can bring to the node's local girth before deciding which check-node should be joined to the current variable node.

Look-Ahead Reverse PEG (PEG-LAR) is proposed in this work as an alternative to the Look-Ahead enhanced PEG. It is very similar to the PEG-LA algorithm as can be seen in 5.

Among the methods described above, the Look-Ahead PEG method is the most costly. In fact it creates many hypothetical graphs before adding each additional edge, as detailed in Chapter 3. Having this added complexity in mind, only one graph was generated with the PEG-LA and PEG-LAR algorithms for each degree distribution and blocklength, while faster methods made it possible to choose the best graph (the one with fewer length-four cycles) out of five independent attempts.

One of the reasons for testing the Look-Ahead Reverse variation is the possibility that the criterion for local girth maximization may not be the best one, since it does not necessarily maximize the graph's girth. The Look-Ahead variation maximizes the largest cycle length involving each new variable node, but not the shortest.

When expanding a tree from an unfinished graph, an edge that completes a cycle will bring no new elements (leafs) to the tree. It can be easily verified that a tree with shorter cycles will therefore grow more slowly than a tree with long cycles. Since the number of nodes is finite and a tree is finished when all the nodes in the graph are reached from the root, the tree with a large girth shall be finished in fewer steps.

We build IRA codes using the five distributions provided by [JKM2000] with block-length $k = \{500, 1000, 2000\}$ and test them on the Additive White Gaussian Noise (AWGN) channel for performance evaluation. With the goal of identifying the possible causes for our results, we also count the number of length-four cycles in each graph, and generate codewords from low weight input messages to have a sample of low weight codewords. The low weight codewords give an insight into the code's distance properties.

We evaluate the performance of these codes through computer simulation. The virtual communication system consists of an encoder, a BPSK modulator, the AWGN channel and a soft-decision decoder that implements the message passing algorithm to obtain a valid codeword from the received block. Block-error probabilities were estimated by repeatedly transmitting encoded blocks of random data until ten error events occurred. An error event is declared when the decoder does not deliver a codeword that is identical to the transmitted block, and it may consist of detected or undetected block-errors.

The decoder declares a decoder failure when even after a maximum allowed number of iterations it cannot confirm a codeword through all parity checks. When the decoder decides towards a codeword that does not match the transmitted block an undetected block-error is declared. Undetected errors are more dangerous than detected errors, thus a code is considered ineffective when undetected errors account for a large proportion of a noticeable count of error events.

Using ten block-error events for each SNR value in the simulation allows for very narrow confidence intervals in the approximated bit-error probabilities. However, at very low bit-error probabilities approaching 10^{-7}, accurate estimations become more time-consuming and fewer samples are collected, when the simulation is timed-out, resulting in more significant error-margins. When ten block-errors are discriminated among detected and undetected block-errors, the individual probability estimation for each category of block-errors become less reliable, so code design is only invalidated on the basis of low minimum distance when undetected errors dominate the error events.

4.2 Degree Distributions

Before comparing the graph construction methods we compare the performances of IRA codes with different degree distributions, constructed with the standard PEG algorithm.

4.2.1 Rate 1/3

We compare the three distributions from [JKM2000] for codes with $k = 1000$ and rate approximately $1/3$ designed under the standard PEG algorithm (PEG-ST). Table 4.2.1 gives the regular check-node degree, the mode

in the edge degree distribution,[2] the lower threshold on $E_b/\mathcal{N}_0$ for successful decoding under belief propagation, and in the bottom rows are the number of length-four cycles and minimum distance of our graphs.

Table 4.1: PEG-ST: $rate \approx \frac{1}{3}$

Distr.	#1	#2	#3
a	2	3	4
mode (λ_i)	$\lambda_6 \approx 0.64$	$\lambda_{13} \approx 0.49$	$\lambda_{27} \approx 0.45$
max. degree	$\lambda_6 \approx 0.64$	$\lambda_{13} \approx 0.49$	$\lambda_{28} \approx 0.02$
Rate	0.333364	0.333223	0.333218
$\left(\frac{E_b}{\mathcal{N}_0}\right)_{thr.}$ [dB]	0.190	-0.25	-0.371
$k = 2000$			
# 4-cycles	0	0	2
d_{min}	32	34	33
$k = 1000$			
# 4-cycles	0	1	7
d_{min}	29	31	31
$k = 500$			
# 4-cycles	0	0	17
d_{min}	25	29	30

In repeated experiments for block length 1500 ($k = 500$), distributions #2 and #3 show consistently tied performance, ranking better than distribution #1 as predicted by density evolution in [JKM2000].

These curves are displayed individually in the Appendix D with the upper error margins. In Figure 4.1, the BER vs. $\frac{E_b}{\mathcal{N}_0}$ curves for IRA codes with $n \approx 1500$ are shown for the three distributions listed in Table 4.2.1.

As shown in Table 4.2.1, the codes built from distributions #2 and #3 were found to have the same minimum distances, but distribution #3 is more prone to forming cycles, due to higher node-degrees. We would assume that distribution #3 would fare better then the others when encoding longer blocks, but for block-lengths $n \approx 1500$ distributions #3 and #2 show similar results.

These results motivated the simulated transmissions using codes built from these same distributions with $n \approx 6000$, which showed distribution #2 outperforming distribution #3 as seen in Figure 4.2. All distributions show non-zero bit error probabilities at channel conditions above the threshold obtained from density evolution. Despite the fact that the performance thresholds assume infinite block-length and, consequently, the expectation that codes for

[2]This is not the same as the node degree distribution, refer to equations A-5 and A-7 in Appendix A for the unique relation between the two.

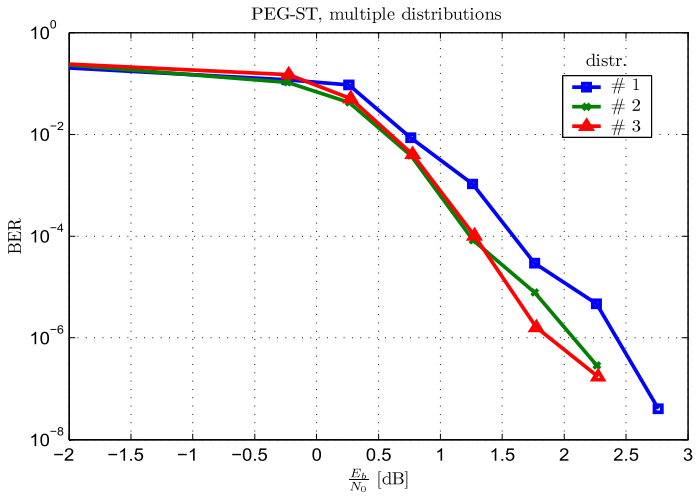

Figure 4.1: Code performance using degree distributions #1, #2 and #3 with $n \approx 1500$

larger blocks using these same distributions would bring better results, the unexpected better performance of distribution #2 when compared to #3 suggests that the PEG-ST algorithm does not fully exploit the potentials of each degree distribution. The results suggest that these codes did not exploit the full potential of these distributions, and motivate the search for better construction methods than the standard PEG algorithm.

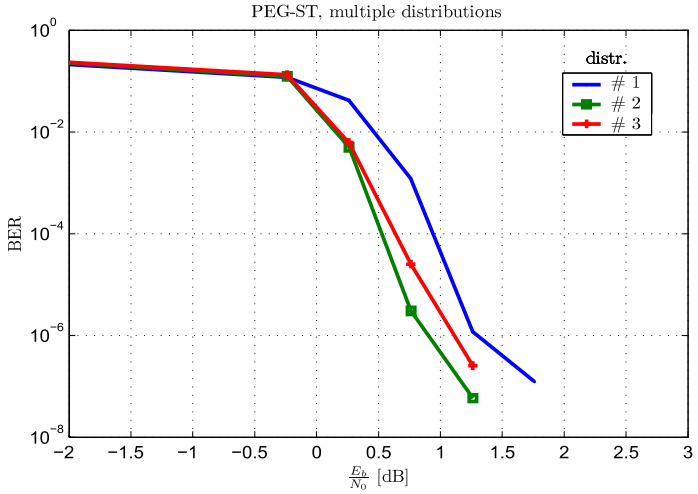

Figure 4.2: Code performance using degree distributions #1, #2 and #3 with $n \approx 6000$

Under the PEG algorithm, length-four cycles tend to involve only the variable nodes of higher degrees, which rely on messages from many additional nodes for decoding. Two such cycles were found in the graph built with underlying distribution #3 and block length $n \approx 6000$, but their influence

on the performance cannot be asserted from these results.

It is possible to explore the distance properties of a code by encoding low-weight messages and plotting the Hamming weight of the resulting codewords. Figure **??** shows one of such plots, where we can observe the minimum weight of the code (in logarithmic scale, polar plot) and the lowest weight of the densest 99% and 95% of these codewords, i.e. the sets containing 95% and 99% of the codewords with highest Hamming weight among those that were generated in this test. The codewords used for this comparison were generated from input messages of Hamming weight 1 and 2.

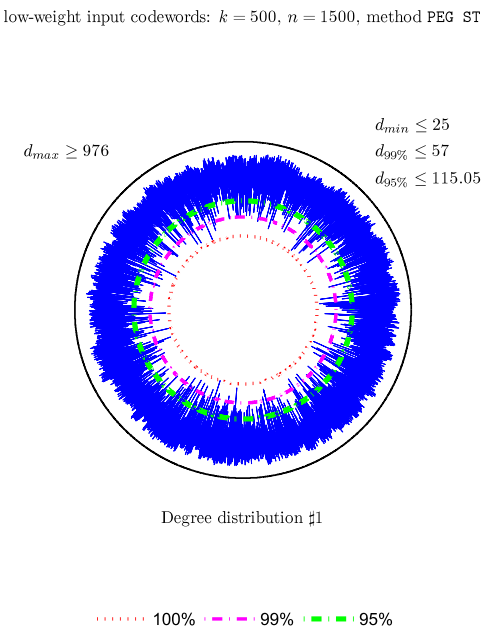

Figure 4.3: Low weight codewords, distribution #1, $n = 1500$

The space between the two innermost dashed circles shows the range of the weights of the 1% least dense codewords (i.e. the ones with lowest Hamming weights). Those are error patterns that may lead to undetected decoding errors at high $\frac{E_b}{\mathcal{N}_0}$ resulting in higher error floors. LDPC codes are not as dependent on good distance properties as older algebraic codes, successfully correcting many error patterns containing a larger number of flipped bits than its minimum distance, but a very low minimum distance would bring high error floors to a code's performance.

The graphic in Figure 4.3 does not show all the information on the distance properties of the graph. An exact notion of the graph's distance properties would require knowledge of all valid codewords, and prohibitive computation time. We can, however, use a histogram of a large random sample of codewords to observe how their Hamming weights are distributed.

Figures 4.4 to 4.6 show that the shape of these histograms resemble a Normal probability density. A large number of low-weight codewords will cause undetected errors at high $\frac{E_b}{N_0}$, therefore it is desirable that the weight histograms (bell-shaped) have large average (μ) and low variance (σ^2).

The histograms obtained for codes with message blocklength $k = 500$ show a clear advantage of distribution #3 over the other two distributions with rate 1/3, since it has the lowest variance and all three have very close Hamming weight average. Distributions #4 and #5 (see histograms in Figures 4.9 and 4.10), on the other hand, require a closer look.

An examination of the interdependence of cycles and the distance properties of a graph through stopping sets can be found in [TJV2003].

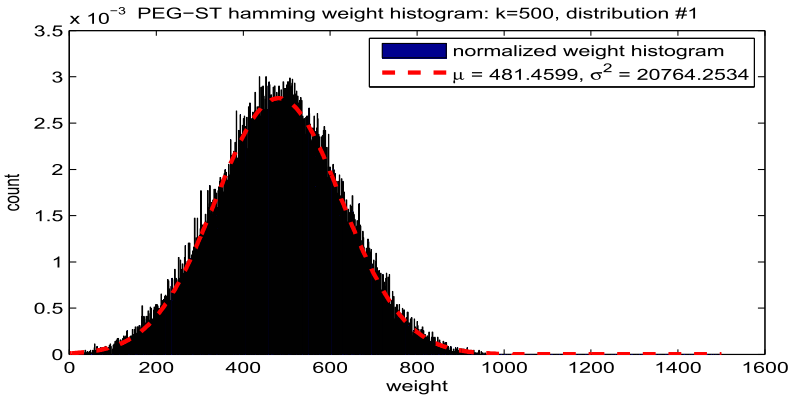

Figure 4.4: histogram of low-weight codewords for distribution #1, $k = 1000$

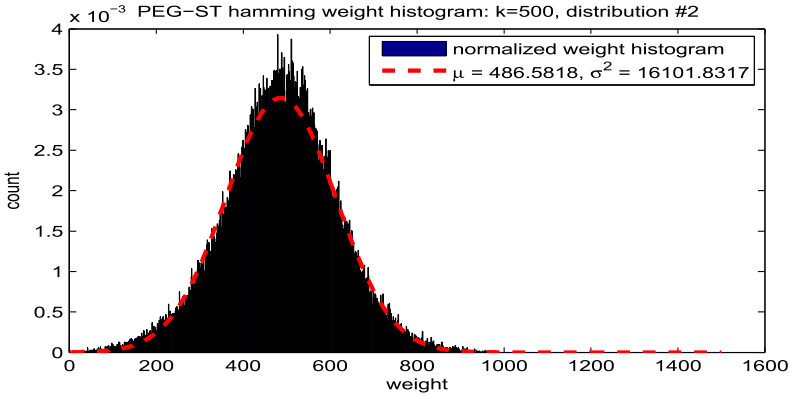

Figure 4.5: histogram of low-weight codewords for distribution #2, $k = 1000$

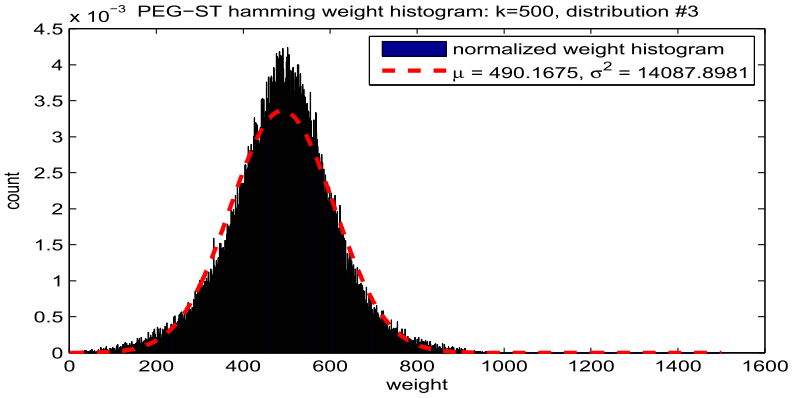

Figure 4.6: histogram of low-weight codewords for distribution #3, $k = 1000$

4.2.2 Rate 1/2

The degree distributions provided for codes with rate approximately $\frac{1}{2}$ are briefly described in table 4.2. The two distributions include higher degrees than the previous three, although the highest degrees are not the mode. For this reason, length-4 cycles become inevitable for these block lengths and may cause greater harm to the performance when using these distributions. Distributions #4 and #5 have the same regular right-degree (a=8), the sole difference being that distribution #4 does not have nodes with degree two.

Table 4.2: PEG-ST k=1000,$rate \approx \frac{1}{2}$

Distr.	#4	#5
a	8	8
mode (λ_i)	$\lambda_{12} \approx 0.33$	$\lambda_7 \approx 0.22$
max. degree	$\lambda_{48} \approx 0.15$	$\lambda_{58} \approx 0.20$
Rate	0.50227	0.497946
$\left(\frac{E_b}{N_0}\right)_{thr.}$ [dB]	0.344	0.266
$k = 2000$		
# 4-cycles	995	2250
d_{min}	19	22
$k = 1000$		
# 4-cycles	2895	3969
d_{min}	12	18
$k = 500$		
# 4-cycles	4203	5298
d_{min}	11	16

Although the decoding thresholds obtained through density evolution

favor distribution #5 by 0.078dB, the absence of degree-two variable-nodes in distribution #4 make the graph less likely to produce weight-two codewords [JKM2000]. The better distance properties of distribution #4 should keep the decoder from deciding towards neighboring codewords, thus achieving better performance. A random search could not detect these codewords, as table 4.2 accuses a greater minimum distance in distribution #5 for all tested block lengths.

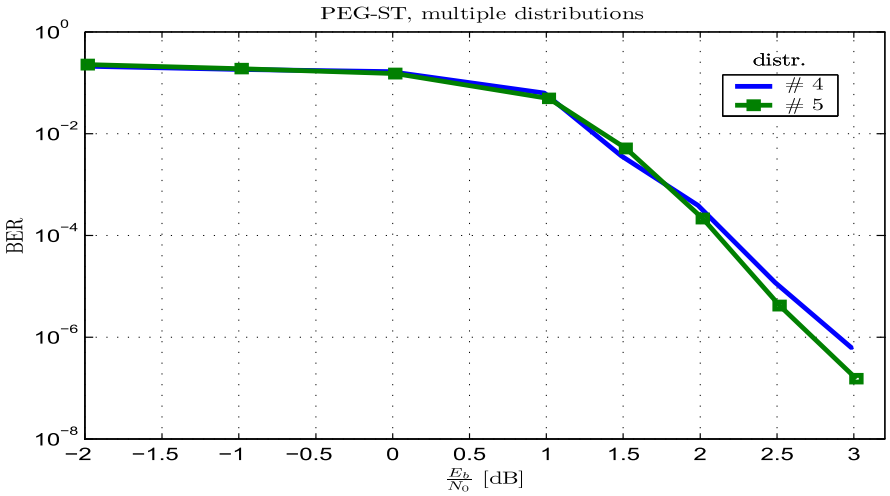

Figure 4.7: Code performance using degree distributions #4, and #5 with $n \approx 1000$

Figure 4.7 shows no advantage for distribution #4 for $n = 1000$ when considering bit errors alone. We can notice, however, that distribution #4 is more protected from undetectable bit-errors comparing Figures D.1 and D.1 (see Appendix D). The **block-error** probability for distribution #4 ($P_e = 4 \cdot 10^{-6}$) is also lower than that of distribution #5 ($P_e = 9 \cdot 10^{-6}$), another consequence of undetected errors that confirms the superiority of distribution #4 in cases where undetected errors are unacceptable.

Figures 4.9 and 4.10 show the Hamming weight histograms of the rate 1/2 codes for $n \approx 1000$. If looked closely (as in Table 4.2.2), the histograms will show an advantage for distribution #5, where the minimum distance is 16 (compared to 11 for distribution #4). However, the code following the distribution #4 did not suffer from undetected errors as did the code built from distribution #5. Therefore, encoding random low weight messages is not an effective heuristic approach for exploring a code's distance properties.

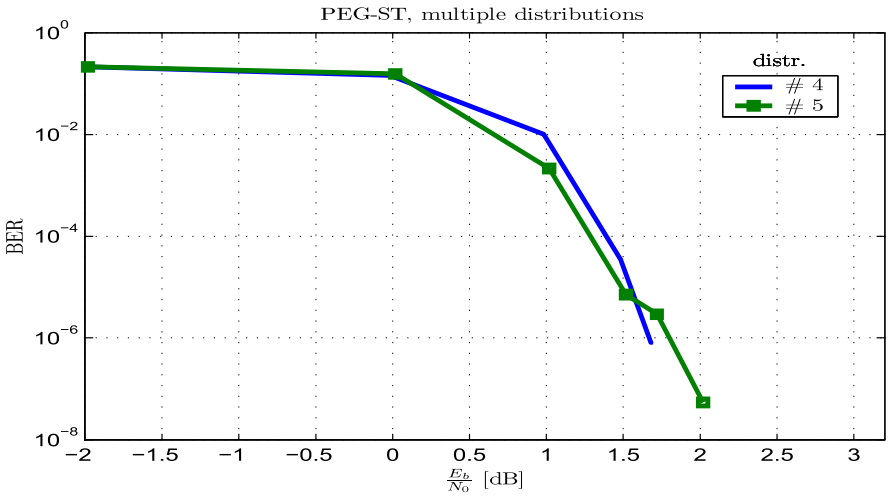

Figure 4.8: Code performance using degree distributions #4, and #5 with $n \approx 4000$

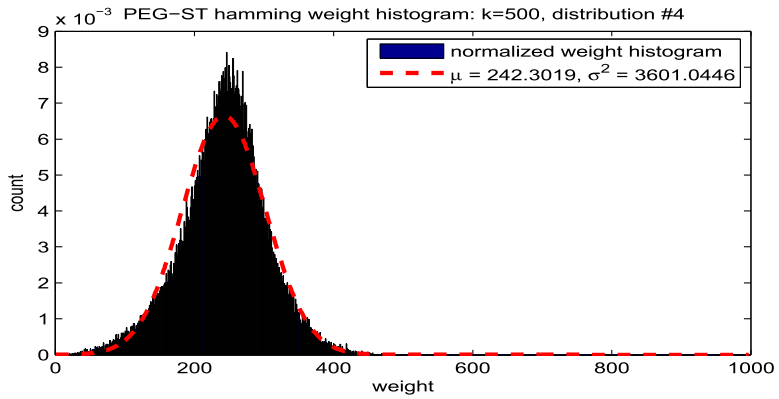

Figure 4.9: histogram of low-weight codewords for distribution #4, $k = 500$

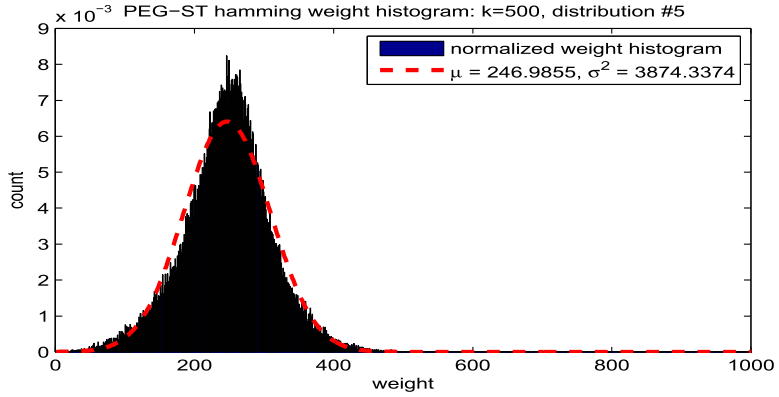

Figure 4.10: histogram of low-weight codewords for distribution #5, $k = 500$

Table 4.3: Count of codewords with the lowest weights on the histograms in figures 4.9 and 4.10.

Ham. weight	11	14	16	20	21	22	23	24	25	26	27	28	29	30
Distr. #4	1	1	0	2	1	0	1	2	0	2	2	3	2	1
Distr. #5	0	0	1	0	1	2	3	4	3	3	2	2	3	3

4.3 Construction Methods

The large number of length-four cycles in the graphs generated from distributions #4 and #5 motivate the use of the Look-Ahead enhanced version of the PEG algorithm for better results. The complexity of the Look-Ahead enhanced version increases quadratically with the block-length, since the algorithm expands a new local graph for each check-node that is eligible for receiving an edge. For practical codes, this imposes a very high cost in computation time.

It is also important to show the cost that is paid in degraded performance when using simpler construction methods. We start with the Level-One variation, the Non-Greedy PEG, and proceed to the look-ahead and look-ahead-reverse enhanced variations.

4.3.1 Level One

This is the most random and intuitive of the discussed methods, and also the poorest as can be seen in 4.11. A coded transmission using an ill connected graph may not even outperform an uncoded transmission, where the power used to transmit the redundant bits is used instead to transmit the raw message symbols through a stronger signal.

Figure 4.11 shows the performance of one code obtained from this method, the graphic shows a random behavior where the bit-error rate does not fall smoothly with growing signal to noise ratio. The rise in errors that can be seen at $\frac{E_b}{N_0} = 2.5$dB may be attributed to error patterns that cannot be corrected by message passing in a poorly connected graph. These error patterns will eventually happen until the SNR is sufficiently high to prevent any errors in during transmission. Therefore, we can say this code is ineffective, and that only avoiding length-four cycles is not enough for obtaining effective error-correcting codes.

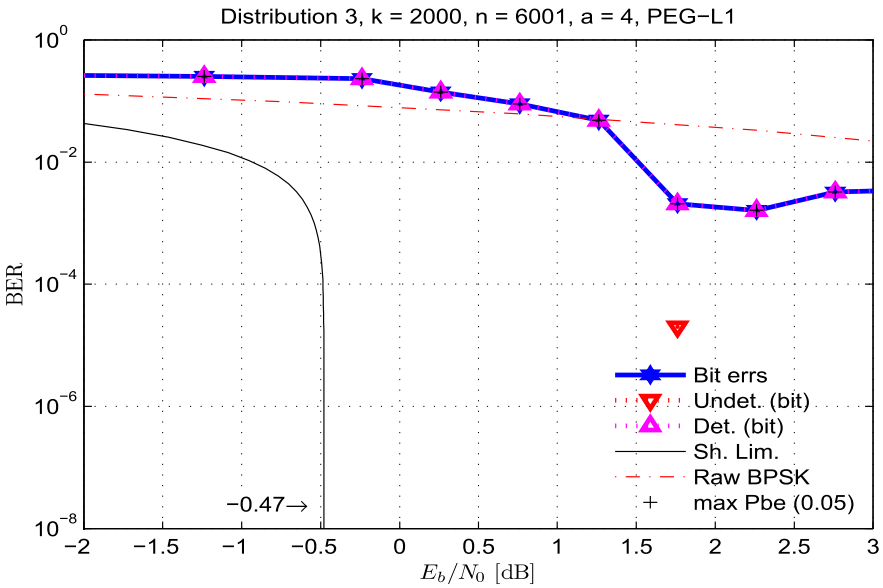

Figure 4.11: Code performance using degree distribution #3 with the PEG Level One algorithm

4.3.2 Non-Greedy PEG

A Non-greedy version of the PEG algorithm does not attempt to maximize the graph's girth. It only prevents the length-four cycles and double edges (identical edges that have no effect on encoding and degrades decoding) while still populating the right-hand side of the graph in a regular manner.

It shows an improvement when compared to the level-one approach, but still far from desirable performance. The performance gains obtained from the additional tree expansion steps in the standard PEG algorithm will pay off in the block lengths used for this work.

The non-greedy version would, however, be useful in a situation where the block length makes the PEG-ST algorithm unpractical and the decoder is required to decode the block in a limited number of iterations (defined by the acceptable latency for the given application). In this case, the maximum number of iterations in the decoder tells how many depth levels the tree should be allowed to expand before allowing a new cycle to be closed. These cycles would not affect the decoder's performance.

The results in Figure 4.13 show one such case where the non-greedy algorithm yields a good code. The low-weight codewords do, however, hamper the performance of the code built from distribution #5.

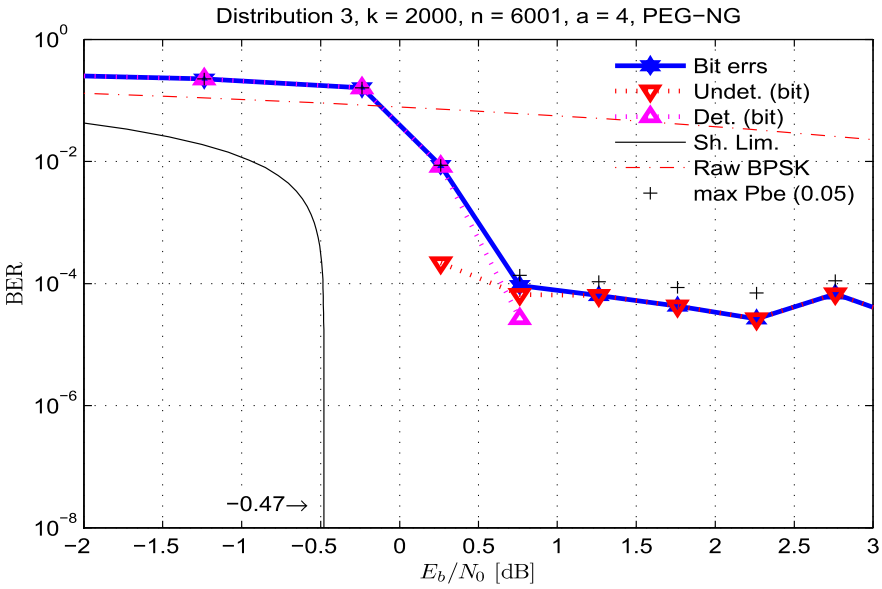

Figure 4.12: Code performance using degree distribution #3 with the non-greedy PEG algorithm

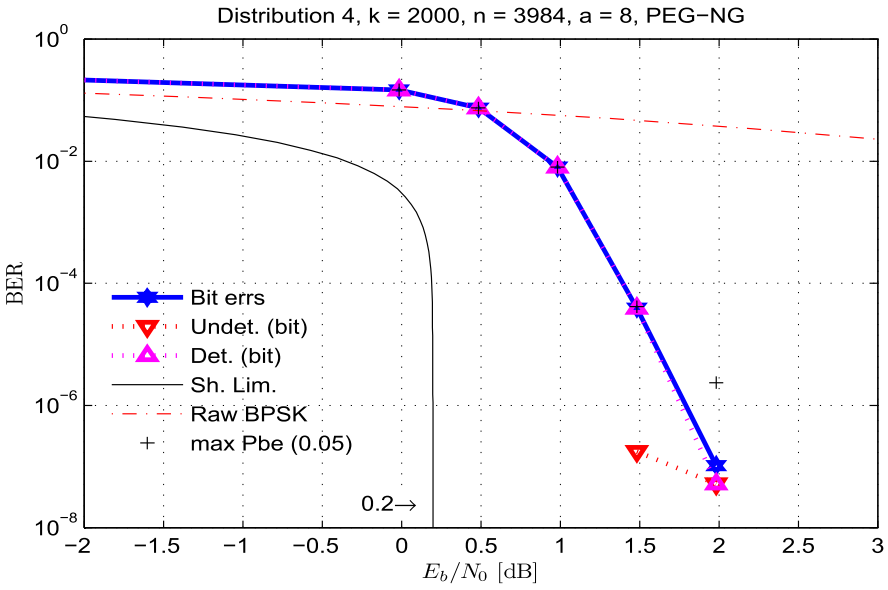

Figure 4.13: Code performance using degree distribution #4 with the non-greedy PEG algorithm

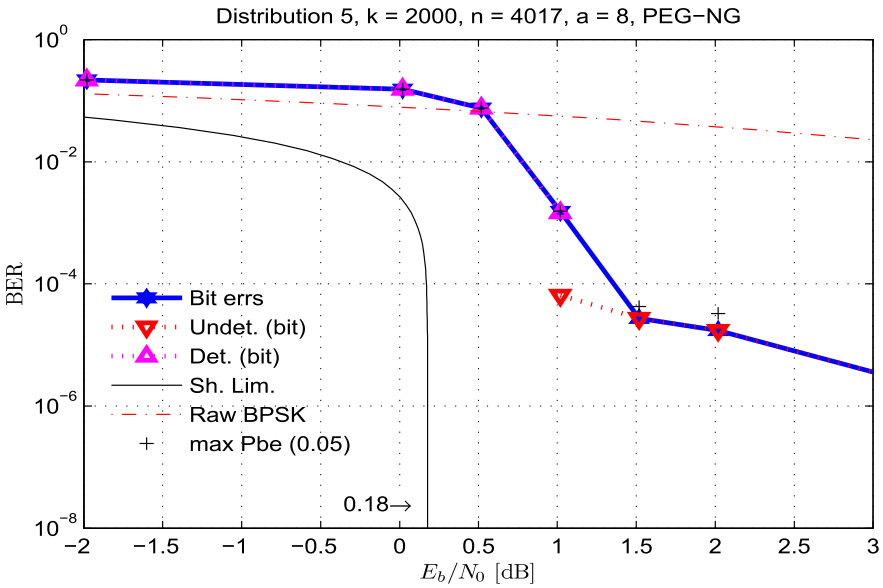

Figure 4.14: Code performance using degree distribution #5 with the non-greedy PEG algorithm

4.3.3
Look-Ahead enhanced PEG

We test the Look-Ahead enhanced versions of the PEG algorithm in its original form (PEG-LA) and with the reversed criterium (PEG-LAR). The results are superposed with the standard PEG in Figures 4.15 and 4.16. The block length $k = 500$ was chosen to illustrate this comparison because it is the one where the need for improved graph construction is more critical.

The criterium for deciding which check-node makes the best candidate for receiving a new edge is also noteworthy. Choosing the hypothesis that leads to the deepest tree may be a misleading strategy since a graph that expands as a tree with many depth levels is a graph with a large maximum cycle-length, but not necessarily a large girth. In fact, graphs with short girth will expand slowly as fewer new check nodes are reached at each subsequent depth level. Graphs with large girth will add more new check-nodes to the tree at each step during the tree expansion, reaching all check-nodes in fewer steps.

As an alternative strategy, the reverse criterium was adopted for a new version which we label PEG-LAR (PEG Look-Ahead reverse). The check node that reaches all others in *fewer* expansion steps is joined with the current variable node. The results of the simulations are in Figures 4.15 and 4.16.

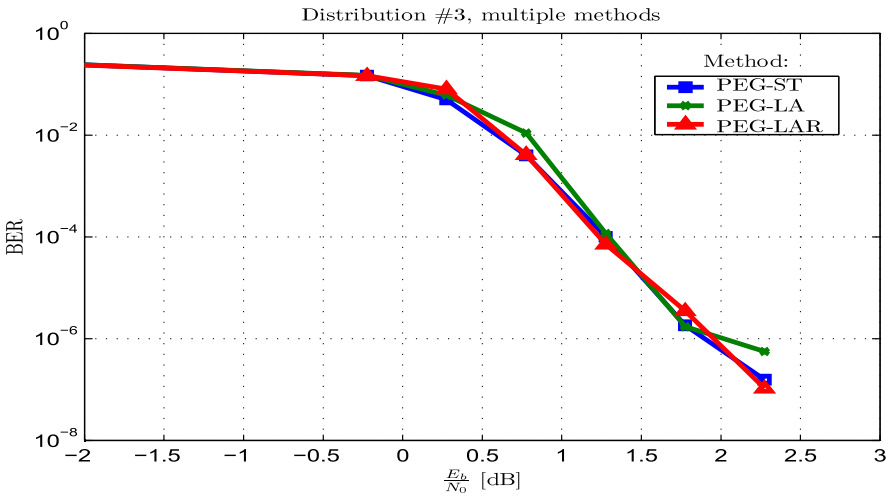

Figure 4.15: Compared performance of methods PEG-ST, LA and LAR, at $k = 500$, distribution #3

It becomes clear from these two plots that, in the specific case of IRA codes, enhanced testing does not bring advantages to the graph construction method. This may not be the case for other classes of LDPC codes. Other distributions or block lengths show very tight advantages for one method or the other without a clear tendency justifying the election of a winner. The

most appropriate of the methods that were tested is, therefore, the standard PEG algorithm.

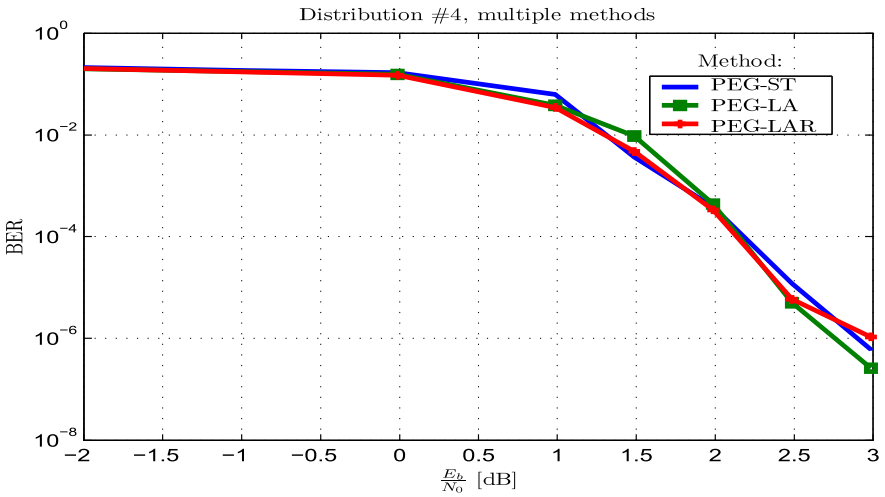

Figure 4.16: Compared performance of methods PEG-ST, LA and LAR, at $k = 500$, distribution #4