Efficient Parametric Decoder of Low Density Lattice Codes

Yair Yona
Dept. of EE-Systems
Tel-Aviv University
Tel-Aviv 69978, Israel
Email: yairyo@eng.tau.ac.il

Meir Feder
Dept. of EE-Systems
Tel-Aviv University
Tel-Aviv 69978, Israel
Email: meir@eng.tau.ac.il

***Abstract*—A new efficient parametric algorithm for implementing the low density lattice codes belief propagation decoder is presented. In the new algorithm the messages passed over the edges are represented by Gaussian parameters lists, and the decoding algorithm uses the low density lattice codes propagation properties in order to group lists efficiently according to a new criteria. The new algorithm attains essentially the same performance as the quantized decoder, proposed in previous work. The new algorithm advantage in comparison to previous works is its smaller storage requirements and its relatively low computational complexity.**

***Index Terms*—Low Density Lattice Codes, Parametric approach, Efficient Decoding**

I. Introduction

Low density lattice codes have been recently presented in [1] and provide an efficient, capacity achieving scheme for coded modulation. These lattice codes are designed directly in the Euclidean space, by using a lattice in which the inverse of its generating matrix is sparse. Interestingly, these codes were shown to have high coding gain, while their "low density" nature led to linear complexity iterative decoding algorithm.

The decoding algorithm presented in [1], while linear in the block length, has still a high computational complexity, and even worse, requires large storage. This is because the messages in the iterative, message passing, algorithm are continuous functions - the PDFs of the continuous codeword samples. In [1], these functions are sampled and quantized resulting in large storage requirements and computational complexity that correspond to the sampling and quantization resolution. Attempts to make the resolution coarser led to degradation in performance. In [2] a parametric representation of the messages has been suggested, which decreases significantly the storage requirements, but still has relatively large linear complexity coefficient. Our work follows this direction and presents an efficient decoding algorithm that uses both the parametric representation and the understanding of the code structure to come up with a very efficient scheme (both in computation and storage) for implementing the belief propagation decoder.

Specifically, in the new algorithm the messages passing over the edges are represented by Gaussian parameters lists. The decoding algorithm uses the low density lattice codes propagation properties in order to group lists efficiently according to a new criteria, based on locating the strongest Gaussians in the lists, and grouping the strongest Gaussians with Gaussians in its surrounding according to second moment matching method. As noted above, the new algorithm advantage is its better storage requirements and the relatively low computational complexity. Comparison between setups that yield the same performance shows that the new algorithm improves the storage requirements by more than two orders of magnitude compared to [1], and improves the computational complexity by more than an order of magnitude compared to [2]. Experimental analysis of the proposed algorithm at different block lengths (up to $n = 100000$), code degree $d = 7$ and with different list lengths shows that the new algorithm attains essentially the same error performance as the quantized decoder of [1] for lists of length $M = 6$ and suffers degradation in performance of $0.1 - 0.2$ db for lists of length $M = 2$.

The outline of the paper is as follows. In section II basic definitions are given. In Section III the iterative decoder algorithm is presented. Section IV introduces method of grouping Gaussian mixture into a single Gaussian. In section V an algorithm of reducing a Gaussian mixture into a smaller Gaussian mixture is presented. The new parametric decoder is defined in section VI. In section VII implementation considerations are presented followed by an analysis of the algorithm efficiency and simulation results in section VIII.

II. Basic Definitions

A. Lattices and Lattice Decoding

An n dimensional lattice in $\mathbb{R}^n$ is defined as the set of all linear combinations of n linearly independent vectors in $\mathbb{R}^n$, with integer coefficients. The matrix G, whose columns consist of the n linearly independent vectors in $\mathbb{R}^n$, is called the generator matrix of the lattice. Each lattice point is constructed from the multiplication $\underline{x} = G\underline{b}$, where $\underline{b} \in \mathbb{Z}^n$. The Voronoi cell of a lattice point, is the set of points in $\mathbb{R}^n$ that are closest to the lattice point. In a squared generator matrix, the Voronoi cell volume equals $|det(G)|$.

The setting in [1] assumes lattice decoding on the unconstrained AWGN channel. The unconstrained AWGN channel required a generalized definition of channel capacity, produced

978-1-4244-4313-0/09/$25.00 ©2009 IEEE

in [3]. When applied to lattices, the channel capacity implies that there exist a lattice G of high enough dimension n that enables transmission with arbitrary small error probability, if and only if $\sigma^2 < \frac{\sqrt[n]{|det(G)|^2}}{2\pi e}$, where σ^2 is the noise variance.

B. Low Density Lattice Codes

LDLC (Low Density lattice Codes) are determined by their non-singular squared generator matrix G. The lattice parity check matrix is defined as $H = G^{-1}$, and in LDLC H is a sparse matrix. Every lattice point $\underline{x}$ satisfies:

$$H\underline{x} = G^{-1}\underline{x} \in \mathbb{Z}^n \tag{1}$$

A "Latin square LDLC" is defined as an LDLC whose parity check matrix H is a Latin square matrix, i.e. a matrix where every row and every column has the same d non-zero values, except for random signs and a possible change of order. The sorted non-zero sequence of these d values, $h_1 \geq h_2 \geq \cdots \geq h_d > 0$, will be referred to as the generating sequence of the Latin square LDLC. In this paper (as in [1] and [2]) we use Latin square LDLC and we normalize the Voronoi cell volume such that $|det(G)| = 1$.

III. Low Density Lattice Codes Iterative Decoder

Using LDLC over the AWGN channel, we have:

$$\underline{y} = \underline{x} + \underline{z} \tag{2}$$

where $\underline{x}$ is the lattice code word, $\underline{z} \sim N(\underline{0}, \sigma^2 I)$, σ^2 is each dimension noise variance and I is the n-dimensional unit matrix.

The LDLC decoder estimates the PDFs: $f_{x_i|\underline{y}}(x|\underline{y}), \quad i = 1 \ldots n$. Analogously to LDPC iterative decoding, each iteration of the LDLC decoder of [1] is composed of 2 steps: passing messages to the check nodes, that represent the parity check equations (a line in the parity check matrix H), and passing messages to the variable nodes that represent elements in the transmitted code word. In each iteration, without loss of generality, variable node l sends d different messages to d different check nodes (the equations where variable node l takes place) and vice versa. For the AWGN channel, the passed messages consist of Gaussian mixtures, where each Gaussian in the mixture can be determined by 3 parameters - mean m, variance V and amplitude a:

$$a \cdot N(x, m, V) = \frac{a}{\sqrt{2\pi V}} e^{-\frac{(x-m)^2}{2V}} \tag{3}$$

Specifically, the LDLC decoding algorithm steps are as follows:

A. Initialization

Without loss of generality we consider variable node l. Each of the d messages are initialized to $N(x, y_l, \sigma^2)$, denoted as the channel observation, where y_l is the $l'th$ element of $\underline{y}$ defined in (2).

B. Check Node

Without loss of generality we consider check node j. We define the variable nodes that take place in check equation j as $x_1, \ldots, x_d$. According to (1) we know that:

$$x_m = \frac{i}{h_m} - \sum_{\substack{k=1 \\ k \neq m}}^{d} \frac{h_k \cdot x_k}{h_m} \qquad m = 1, \ldots d \tag{4}$$

where i is an integer. We also define:

$$r_m = \sum_{\substack{k=1 \\ k \neq m}}^{d} \frac{h_k \cdot x_k}{h_m} \qquad m = 1, \ldots d \tag{5}$$

In [1] it has been shown that the $d-1$ random variables that take place in the right side of equation (4) can be referred as statistically independent random variables, hence the PDF of r_m in (5) is a convolution between the incoming messages after expanding/streching according to the coefficients $h_1, \ldots, h_d$. As the check equation integer solution i in (4) is unknown, the PDF of r_m is periodically expanded to period of $\frac{1}{h_m}$ in order to yield the check node message sent to x_m. In this way, x_m message takes into account all possible solutions. As the PDF of each incoming message is a Gaussian mixture, the PDF of r_m is also a Gaussian mixture and so is the message of x_m. Each Gaussian in the message of x_m consists of the convolution between $d-1$ different Gaussians from $d-1$ different incoming messages, and a shift of $\frac{l}{h_m}$, where l can have any value in $\mathbb{Z}$. Denote as, $m_k, V_k, a_k \; k = 1, \ldots, d-1$, the means, variances and amplitudes of certain $d-1$ Gaussians from $d-1$ different incoming messages that take place in the convolution, and a shift of $\frac{l}{h_m}$. The Gaussian parameters of the corresponding Gaussian in the message of x_m will be: $m_x = \frac{l}{h_m} - \sum_{k=1}^{d-1} \frac{h_k \cdot m_k}{h_m}$, $V_x = \sum_{k=1}^{d-1} \frac{h_k^2 \cdot V_k}{h_m^2}$ and $a_x = \prod_{k=1}^{d-1} a_k$.

C. Variable Node

Without loss of generality we consider variable node j. We denote the d check nodes that send messages to variable node j as $c_1, \ldots, c_d$ and their periodic messages as $p_1(x), \ldots, p_d(x)$ (the messages are Gaussian mixtures). The estimated PDFs are:

$$\hat{f}_{m,x_j|\underline{y}}(x|\underline{y}) \propto \prod_{\substack{k=1 \\ k \neq m}}^{d} p_k(x) \cdot N(x, y_j, \sigma^2) \qquad m = 1, \ldots, d \tag{6}$$

The periodic messages, sent by the check nodes, consist of Gaussian mixtures. The channel observation consists of a single Gaussian. Multiplication of Gaussian mixtures yields a Gaussian mixture. Hence, the estimated PDF $\hat{f}_{m,x_j|\underline{y}}(x|\underline{y})$ is also a Gaussian mixture. Each Gaussian in $\hat{f}_{m,x_j|\underline{y}}(x|\underline{y})$ consists of the multiplication between $d-1$ different Gaussians from $d-1$ different incoming messages, and the channel observation. Denote as, $m_k, V_k, a_k \; k = 1, \ldots, d$, the means, variances and amplitudes of certain $d-1$ Gaussians from $d-1$ different messages, and the channel observation. The Gaussian

parameters of the corresponding Gaussian in $\hat{f}_{m,x_j|\underline{y}}(x|\underline{y})$ will be:
$V_{\hat{f}} = \frac{1}{\sum_{k=1}^{d}\frac{1}{V_k}}$, $m_{\hat{f}} = V_{\hat{f}} \cdot \sum_{k=1}^{d}\frac{m_k}{V_k}$ and
$a_{\hat{f}} = \frac{e^{-\frac{V_{\hat{f}}}{2}\sum_{k=1}^{d}\sum_{j=k+1}^{d}\frac{(m_k-m_j)^2}{V_k\cdot V_j}}}{\sqrt{(2\pi)^{d-1}V_{\hat{f}}^{-1}\prod_{k=1}^{d}V_k}} \cdot \prod_{k=1}^{d} a_k$.

D. Final Decision

Consider the same definitions as in subsection III-C. In this case: $\hat{f}_{x_j|\underline{y}}(x|\underline{y}) \propto \prod_{k=1}^{d} p_k(x) \cdot N(x, y_j, \sigma^2)$ where $\hat{x}_j = argmax_x \hat{f}_{x_j|\underline{y}}(x|\underline{y})$ and $\hat{\underline{b}} = \lfloor H \cdot \hat{\underline{x}} \rceil$.

The number of Gaussians in the messages grows exponentially as a function of the number of iterations. In order to attain practical parametric decoding, the Gaussians in the messages need to be grouped into a constant number of Gaussians M.

IV. Grouping a Gaussian Mixture Into a Single Gaussian

In order to have a constant parametric list length, we would like to group certain chosen Gaussians into a single Gaussian. The chosen Gaussians constitute a Gaussian mixture:

$$GM(x) = \sum_{k=1}^{L} a_k \cdot N(x, m_k, V_k) \tag{7}$$

$$\bar{GM}(x) = \sum_{k=1}^{L} b_k \cdot N(x, m_k, V_k) \qquad b_k = \frac{a_k}{\sum_{l=1}^{L} a_l} \tag{8}$$

where $\bar{GM}(x)$ is the normalized Gaussian mixture. We would like to approximate $\bar{GM}(x)$ with the Gaussian that minimizes the Kullback-Leibler divergence (D) between them. Hence, we would like to find:

$$(\hat{m}, \hat{V}) = argmin_{m,V} D(\bar{GM} || N(x, m, V)) \tag{9}$$

The Gaussian that minimizes the divergence in (9) is the second moment matched Gaussian, i.e. the Gaussian with the same mean and variance as $\bar{GM}(x)$. Thus, the Gaussian that approximates (7) is:

$$\hat{GM}(x) = \sum_{k=1}^{L} a_k \cdot N(x, \hat{m}, \hat{V}) = \hat{a} N(x, \hat{m}, \hat{V}) \tag{10}$$

and the Gaussian mixture approximation parameters are:
$\hat{m} = \sum_{k=1}^{L} b_k \cdot m_k$, $\hat{a} = \sum_{k=1}^{L} a_k$ and $\hat{V} = \sum_{k=1}^{L} b_k \cdot V_k + b_k \cdot (m_k - \hat{m})^2$.

V. Gaussian Mixture Reduction

We would like to reduce a Gaussian mixture with L Gaussians as defined in (7) to a Gaussian mixture with M Gaussians at most ($M < L$), using the method described on section IV. We suggest 2 algorithms that yield the same output. The difference between the algorithms reflects in the computational complexity. The first algorithm finds which Gaussians to consolidate in a straight forward fashion, while the second algorithm sorts the lists of Gaussians and only then begins the consolidation process. The different approaches have different complexities.

A. Straight Forward Algorithm

The Gaussian mixture of length L can be represented by 3 parametric lists of length L that contain each Gaussian mean, variance and amplitude. We define another list of length L, named "amplitude to s.t.d ratio" list (s.t.d stands for standard deviation), that contains the ratio $\frac{a_k}{\sqrt{V_k}}$, $k = 1, \ldots, L$. Each element in this list is proportional to the coefficient of the exponent in the Gaussian defined in (3). We refer to those lists as the "long lists". The reduced Gaussian mixture of length M, is represented by 3 parametric lists of length M, in a similar manner to the long lists. We refer to the reduced Gaussian mixture lists of length M as the "short lists".

step 1: Initialize the long list length. $ListLength = L$.

step 2: Choose the Gaussian with the strongest value in the amplitude to s.t.d ratio list. Without loss of generality, assume that this Gaussian mean, variance and amplitude are m_k, V_k, a_k respectively.

step 3: Define a range of length $2A$ around m_k:

$$RANGE = \{x | m_k - A \leq x \leq m_k + A\} \tag{11}$$

Go over the Gaussians means in the long lists m_l, $l = 1, \ldots, ListLength$, group the Gaussians that satisfy $m_l \in RANGE$ into a single Gaussian according to (7)-(10). Add the estimated single Gaussian mean, variance and amplitude parameters to the short lists.

step 4: Assume that S Gaussians from the long lists were grouped on *step 3*. Erase from the long lists those S Gaussians. Update the long lists length $ListLength = ListLength - S$.

step 5: Repeat *steps 2-4*, either until $ListLength = 0$ or M times.

In the end of the process, the Gaussian mixture with L Gaussians is reduced to a Gaussian mixture with M Gaussians at most; The computational complexity of this algorithm is $O(L \cdot M)$.

B. Sort Algorithm

Build 4 pointer lists of length L (the long lists), *MeanList*, *VarianceList*, *AmplitudeList* and *AmplitudeStdRatioList*, for the mean, variance, amplitude and amplitude to s.t.d ratio of the Gaussian mixture. Each element in each list points to the elements that come before and after it in the list. Also each element in each list points to elements in the other lists that correspond to the same Gaussian. We also define m_l as the mean value in the $l'th$ element of *MeanList*.

step 1: Update the long lists length $ListLength = L$.

step 2: Sort *MeanList* and *AmplitudeStdRatioList* in descending order. Now the mean and amplitude to s.t.d lists are sorted.

step 3: Choose the strongest element in *AmplitudeStdRatioList* (actually this is the first element in the sorted list). Without loss of generality, assume that the strongest element in *AmplitudeStdRatioList* points to element k inside *MeanList*.

step 4: Mark element k inside *MeanList*. The mean value in the $k'th$ element of *MeanList* is m_k, define a $2A$ range around m_k as in (11).

step 5: Initialize $k_u = k_d = k$.

step 6: $k_u = k_u + 1$. If $m_{k_u} \in RANGE$, mark element k_u in *MeanList* and repeat *step 6*. If $m_{k_u} \notin RANGE$ or we reached the end of the list, continue to *step 7*.

step 7: $k_d = k_d - 1$. If $m_{k_d} \in RANGE$, mark element k_d in *MeanList* and repeat *step 7*. If $m_{k_d} \notin RANGE$ or we reached the beginning of the list, continue to *step 8*.

step 8: Assume that S elements inside *MeanList* have been marked. Take the S marked mean values and its corresponding variances and amplitudes and group the Gaussians they represent into a single Gaussian according to (7)-(10). Add the estimated single Gaussian mean, variance and amplitude parameters into the short lists.

step 9: Extract the marked elements from *MeanList* and its corresponding values from the rest of the long lists. Update the long lists length $ListLength = ListLength - S$. The sorted lists remain sorted (As every subset of a sorted list is also a sorted list).

step 10: Repeat *steps 3-9*, either until $ListLength = 0$ or M times.

Note that the computational complexity of *step 2* is $O(L \log_2 L)$, and the computational complexity of *steps 3-10* is $O(L)$ (going over the L elements in the list). Hence, the computational complexity is dominated by $O(L \log_2 L)$.

As aforementioned, both algorithms yield exactly the same output. The complexity depends on L and M. Given certain M and L we choose the algorithm that has smaller complexity.

VI. Parametric LDLC Decoder

We now incorporate the reduction method developed in section V into the LDLC decoder. In the parametric algorithm the data-base of the passed messages consists of lists of means, variances and amplitudes. Both in the check nodes and the variable nodes, there are $n \cdot d$ lists, each of length M. M represents the number of Gaussians in each message.

A. Variable Node

From (6) we can see that each variable node message consists of multiplication of $d-1$ messages and the channel observation. Without loss of generality we consider variable node j. We denote the Gaussian mixture reduction algorithm defined in section V as $GMReductionAlg(GM(x), M)$, where $GM(x)$ is the Gaussian mixture that needs to be reduced, and M is the maximum number of Gaussians in the reduced Gaussian mixture. The algorithm returns the reduced Gaussian mixture with M Gaussians at most. In order to calculate the variable node messages we use the Forward-Backward algorithm described in [2].

For $i = 2, \ldots, d$, the forward step:

$\hat{FW}_1(x) = N(x, y_j, 2\sigma^2)$

$FW_i(x) = \hat{FW}_{i-1}(x) \cdot p_{i-1}(x)$

$\hat{FW}_i(x) = GMReductionAlg(FW_i(x), M)$.

The backward step:

$\hat{BW}_d(x) = N(x, y_j, 2\sigma^2)$

$BW_{d-i+1}(x) = \hat{BW}_{d-i+2}(x) \cdot p_{d-i+2}(x)$

$\hat{BW}_{d-i+1}(x) = GMReductionAlg(BW_{d-i+1}(x), M)$.

Combining both steps, for $l = 1, \ldots, d$, the variable node messages:

$FWBW_l(x) = \hat{FW}_l(x) \cdot \hat{BW}_l(x)$

$\hat{f}_{l,x_j|\underline{y}}(x|\underline{y}) \propto GMReductionAlg(FWBW_l(x), M)$.

B. Check Node

Calculating r_m : The calculation of the PDF of r_m in (5) requires $d-2$ convolutions. We calculate the PDF using the Forward-Backward algorithm in a similar manner to the method described in VI-A.

Replication: The check node messages in the LDLC decoder are periodic. The periodic extension requires infinite number of replications of each Gaussian in the PDF of r_m. In the parametric algorithm we would like to take a finite number of replications, K, instead of the infinite periodic extension. Without loss of generality let us consider the check node messages that are sent to variable node j. Since we do not know the transmitted code word, for every Gaussian in every message that arrives to variable node j, we take the K replications that are closest to y_j. The replications affect the Gaussians mean values and produces lists of length $K \cdot M$. Such method resembles the behavior in the quantized decoder, in which after the multiplication with the channel observation, as a result of the fact that the channel observation function is quantized, only the closest replications to the channel observation mean y_j remains.

Taking a finite number of replications increases the error probability due to the fact that the tail of the Gaussian noise may not be taken into account. In order to attain error probability that asymptotically converges to zero, K should increase asymptotically as well.

C. Final Decision

In variable node j, in order to find $\hat{x_j}$ we need to find the argument that maximizes $\hat{f}_{x_j}(x|\underline{y})$, and so we need to calculate its quantized function according to the parameters lists. We define a range of length J around y_j with resolution of Δ. We calculate the Gaussian mixture in this range according to the parameters lists.

Note, the computational complexity of calculating the quantized PDF directly can be avoided by estimating $\hat{x_j} = E_{\hat{f}}(x)$. In this case a problem can occur in estimating the correct lattice point, in cases where there is more than 1 hypothesis with similar probability at different lattice points. Such a decision rule can increase the symbol error rate , but we did not see significant degradation in the simulation.

VII. Implementation Considerations

In order to avoid numerical instabilities, we define a minimum value to the variance. We define it as $MinimumVar = \alpha \cdot \sigma^2$. In the variable nodes, a certain Gaussian with variance V, receives a new variance of $MAX(MinimumVar, V)$.

One of the strongest properties of the LDLC is the duality between the variable nodes and the check nodes. For instance, while the variable node performs product over the input messages, the check node performs convolution. In a similar manner, while setting the ranges length to group Gaussians

as defined in (11), we distinguish between the range length in the variable node and the range length in the check node. We define the range length to group Gaussians in the variable node as $VariableNodeRangeLength$ and the range length in the check node as $CheckNodeRangeLength$. Those ranges correspond to A, defined also in (11).

In the check node, since the convolution can cause relatively large changes in the Gaussians means and enables the convergence to the correct code word, it is highly important to keep the uniqueness of each Gaussian after the convolution. Hence, we choose $CheckNodeRangeLength$ to be relatively small. In the variable node, each multiplication is done between d Gaussians. Consider a Gaussian with very small variance, and mean value m which is very close to a certain lattice point, we shall refer to such Gaussian as hypothesis. Also suppose that this Gaussian is multiplied with $d-1$ Gaussians with relatively large variances (this is the LDLC typical behavior). The multiplication yields Gaussian with mean value that is close but not necessarily equal to m. Considering all the multiplications the small variance Gaussian takes place in, we receive Gaussians with mean values that spread around m that support the same hypothesis and strengthen it. We would like to take a relatively large range length that will group them and will enable to sustain the hypothesis. Hence we choose $VariableNodeRangeLength$ to be relatively large. Since the mean values of those Gaussians are relatively close to m, such grouping would not lead to a significant bias in the lattice code word estimation.

We would like to emphasize the tradeoff between the number of Gaussians, M, and the range length A. Consider the value $TotalRange = M \cdot A$. First we increase the number of Gaussians M, while reducing A such that $TotalRange$ remains constant. In this case, the algorithm computational complexity increases. The algorithm "resolution" also increases and the algorithm gives better approximation to the theoretic algorithm. Now we decrease the number of Gaussians M while increasing A. In this case, the computational complexity decreases. On the other hand the increase in the chance of grouping several hypothesis together, can result in the loss of the correct hypothesis and degradation in the performance.

VIII. Storage, Complexity and Simulation Results

In both the check nodes and the variable nodes, there are $n \cdot d$ lists of length M. The storage requirement is $O(n \cdot d \cdot M)$. The storage requirement is similar to the storage requirement in [2]. In [1], the storage requirement was $O(n \cdot d \cdot \frac{L}{\Delta})$, where L is the quantized PDF range length and Δ is the resolution. If we take $M = 6$, $L = 4$ and $\Delta = \frac{1}{256}$, we can see a significant improvement in the storage requirement in comparison to the quantized decoder (improvement by a factor of over 100).

In the parametric algorithm the complexity is dominated by the Gaussian mixture reduction algorithm described on section V. In the straight forward algorithm, described on subsection V-A, the computational complexity is $O(n \cdot d \cdot t \cdot K \cdot M^3)$, where M is the number of Gaussians in each list, K is the number of replications and t is the number of iterations. In the sort algorithm, described on subsection V-B, the computational complexity is $O(n \cdot d \cdot t \cdot K \cdot M^2 \log_2(K \cdot M^2))$. For small M the straight forward algorithm is more efficient and for large M the sort algorithm is more efficient. The computational complexity of the algorithm suggested in [2] is $O(n \cdot d \cdot t \cdot K^2 \cdot M^4)$, and so the new algorithm significantly reduces the complexity (for the same values, by an order of magnitude). The computational complexity of the quantized algorithm suggested in [1] is dominated by $O(n \cdot d \cdot t \cdot \frac{1}{\Delta} \cdot \log_2(\frac{1}{\Delta}))$. While the complexity is not expressed by the same terms, for typical values that yield the same performance there is also an order of magnitude improvement compared to the quantized decoder.

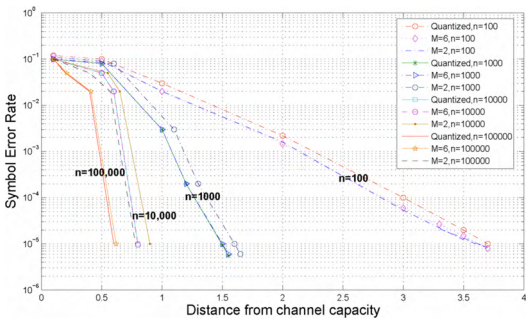

Fig. 1. Symbol Error Rate for different list lengths and block lengths

The simulation results for $M = 2/6$, $d = 7$ ($d = 5$ for $n = 100$), $K = 3$, ranges length (correspond to A defined in (11)) of $VariableNodeRangeLength = 0.2$, $CheckNodeRangeLength = 0.05$ and $MinimumVar = 0.03 \cdot \sigma^2$ are presented in figure 1. We used the same generating sequence as in [1]. The zero code word was used and maximum amount of 200 iterations was allowed. For $M = 6$ the performance is identical to the quantized decoder performance. For $M = 2$ and $VariableNodeRangeLength = 0.6$, there is a slight degradation that varies between $0.1 - 0.2$ db (no degradation for $n = 100$).

IX. Conclusion

In this work we presented a new parametric algorithm for the LDLC decoder. The new algorithm was shown to be more efficient than algorithms suggested in previous works. We believe that LDLC can suit communication systems with high spectral efficiency.

X. Acknowledgment

Support and interesting discussions with Naftali Sommer are gratefully acknowledged.

References

[1] N. Sommer, M. Feder, and O. Shalvi, "Low Density Lattice Codes" submitted to *IEEE Transactions on Information Theory*.

[2] B. Kurkoski, and J. Dauwels, "Message-Passing Decoding of Lattices Using Gaussian Mixtures", IEEE ISIT 2008.

[3] G. Poltyrev, "On Coding Without Restrictions for The AWGN Channel", *IEEE Trans. Inform. Theory*, vol. 40, pp. 409-417, Mar. 1994.