Mathematical Optimization Based Channel Coding: Current Achievements and Future Challenges

Michael Helmling[#], Stefan Scholl[*], Akın Tanatmis[#]

[*]*Department of Electrical and Computer Engineering*
[#]*Department of Mathematics*
University of Kaiserslautern, 67653 Kaiserslautern, Germany
{helmling, tanatmis}@mathematik.uni-kl.de
scholl@eit.uni-kl.de

***Abstract*— Channel coding and mathematical optimization— two omnipresent branches of science which heavily influence our everyday life, which is certainly unimaginable without the epochal achievements of each of the two disciplines since they affect nearly every communication system as well as every transportation, manufacturing, and organization process. The following report is dedicated to some of the achievements of a research project decisively influenced by a cooperative interplay of these two disciplines.**

I. Channel Coding

Channel coding is an important technique for the correction of transmission errors that is used in nearly every communication system today. It is used in mobile phones as well as in satellite communication, navigation systems, storage devices like hard disks, CDs and also in internet or broadcast applications. Let us first review what channel coding is and why we need it. To understand the problems channel coding solves, we will have a look at the old days. As an example we take a communication system without channel coding that becomes more and more obsolete: analog TV.

Television broadcasting is a form of one way communication. On the one hand we have a transmitter, which is usually placed on top of a mountain and transmits radio waves with high power to reach an area as big as possible. On the other hand we have the receivers, ordinary TV sets, in our households, that pick up the radio waves.

If the connection between transmitting antenna and receiving antenna is of good quality, we a see clear picture and hear a comfortable sound. But what happens if the quality of the connection is not so good? Let us assume our receiver is placed far away from the transmitter, behind a mountain, in the cellar or simply in bad weather, so that the reception quality drops. Then the picture is distorted or, even worse, the whole picture bounces up and down. Also the sound might be clicking and crackling, disturbing the viewers enjoyment. The TV signal is noisy or distorted.

Figure 1 shows such a situation where we have two TV watchers, A and B. Whereas A is close to the transmitter and receives a clear picture, B is not in such a good position. He has worse receiving conditions due to serveral obstacles.

The goal of new digital TV broadcasting systems is to make picture and sound more resistant to distortion and noise. In

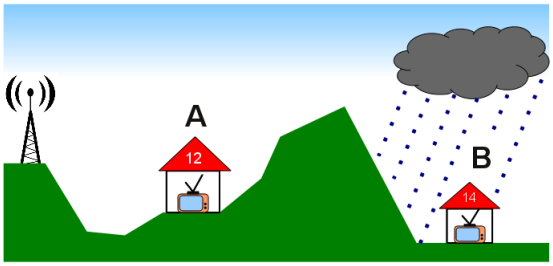

Fig. 1. TV transmitter and two receivers

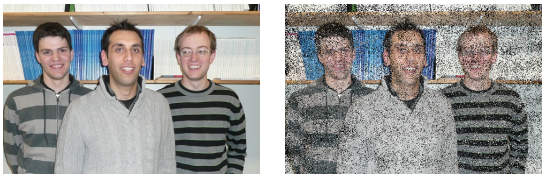

Fig. 2. TV pictures from A and B

contrast to the presented analog system, there are additional possibilities for techniques to improve the link from the transmitter to the TV set at home. One of them is channel coding. Channel coding is a technique to detect and correct occuring errors. But how is that possible?

Whenever a signal is transmitted in modern digital communication systems, the data is transmitted in form of bits or bitstreams, respectively. To allow error correction on the receiver side, we have to make some preprocessing before transmission, called *encoding*. To encode the data, the bitstream is first cut into small blocks. At the end of every block some extra bits, called parity bits, are added as shown in Figure 3. The k data bits together with the r parity bits form a vector of n bits which is called codeword the $\mathbf{c}$.

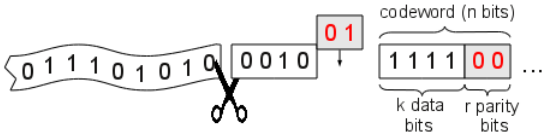

Fig. 3. A bitstream is being encoded

During encoding the additional parity bits are calculated

from the data bits. The "way" we calculate the parity bits is called *channel code* and determines the performance of the channel coding system. Since data is processed block by block, the channel codes described here are called *linear block codes*. A linear block code is defined by a binary $r \times n$ matrix H, so that every bitvector $\mathbf{c}$ that fullfills

$$H \cdot \mathbf{c} = 0 \tag{1}$$

(where the calculation takes place modulo 2) is a codeword. H is called *parity-check matrix*. By $\mathcal{C} = \{\mathbf{c} \in \{0,1\}^n : H \cdot \mathbf{c} \equiv 0 \bmod 2\}$ we denote the set of all codewords [1]. Finding a good channel code with its matrix H is non-trivial and subject to research since decades.

After encoding the data it is sent to the receiver. During transmission some bits may be distorted and change their values.

The actual correction of corrupted data takes place in the receiver by a device called channel decoder, that tries to "repair" the incoming blocks. For this purpose the decoder uses the previously added parity bits. In many cases this technique works well and many errors can be successfully corrected.

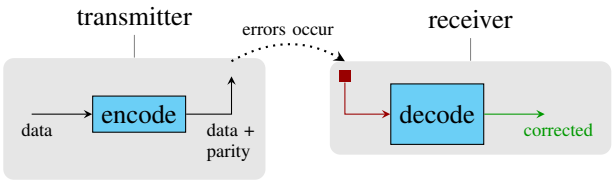

Fig. 4. Encoding, transmission, and decoding of data

However there is a probability that the decoder cannot correct an erroneous block. This remaining error probability is called frame error rate (FER) and is dependent on the signal-to-noise ratio (SNR), which is the signal power divided by the noise power at the receiver. The FER is an indicator for the quality of the channel coding system. It is measured with Monte-Carlo simulation on a computer and visualized with a special chart, shown in Figure 5. This type of chart has a typical shape. For weak signals, that is, low SNR, the FER is very high (close to 1) and drops with increasing signal strength. In Figure 5 there are two curves, the blue one represents a system without channel coding, the red one corresponds to the same system with channel coding. One can easily see that the channel coding provides significantly less errors. The lower a curve lies in the performance chart, the better the coding system works. In fact there are many parameters that influence the performance of channel coding, but the performance chart allows for a good comparison.

In practical applications different error probabilities are necessary. For audio signals, like in a mobile, a minimal error probability 0.01 is required, otherwise crackling or interruptions make a call uncomfortable. For communications through optical fiber error rates of less than 10^{-12} may be necessary for flawless operation.

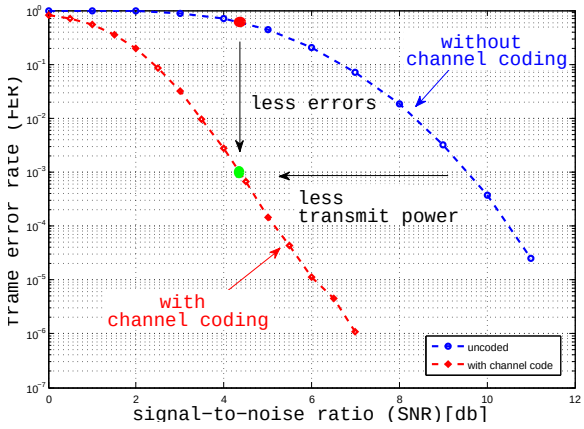

Fig. 5. FER for a system without channel coding and one using channel coding

Using channel coding we can handle more weak signals without any loss of quality or provide much more reliable data for a given signal strength. This advantage can be translated into equipment cost reduction, for example smaller antennas, cheaper electronics or smaller batteries.

We now have seen which processing steps in channel coding are necessary, namely encoding and decoding. The decoder is much more complex than the encoder and is the main part of a channel coding system. So now we will have a closer look at the decoding process.

In channel coding the data is processed in blocks of n bits consisting of k data bits and $r = n - k$ parity bits. Because the parity bits are calculated from the data bits, they are fully dependent on the data bits. Therefore there are only 2^k possible codewords, although there are 2^n possible bit combinations. When a erroneous codeword is received, the channel decoder checks which one of the 2^k valid codewords is most similar to the received block. This process is called maximum likelihood (ML) decoding and is the optimal way of decoding [1]. However, because all 2^k codewords need to be tested, this procedure is applicable only for small values of k. With increasing k the complexity grows exponentially. Since, in practice, we deal with pretty large blocks (for example $k = 100$ or $k = 10000$), the ML decoding procedure of going through all 2^{100} or even 2^{10000} codewords is simply not possible to calculate in an imaginable amount of time.

To overcome this problem engineers often use heuristics or suboptimal algorithms for the decoding procedure. These algorithms are very often capable of finding the correct solution, but with much less complexity. In some cases, however, the suboptimal algorithms decide for a wrong solution. This leads to a small degradation in performance, which one has to accept if the problem should be feasible.

Only if the complexity is low enough, the decoding algorithm can be implemented on a chip for use in practical applications. For many of modern codes, algorithms with a good tradeoff between performance and complexity are known. Moreover, for many used channel codes only the

performance with suboptimal decoding algorithms is known—not the one of optimal ML decoding.

From a mathematicans point of view there exist other means of solving the ML decoding problem than comparing all 2^k codewords to the received one. The decoding can be formulated as follows.

Assume that at the receiver a vector $\tilde{\mathbf{c}}$ was received. It will check all possible sent codewords $\mathbf{c}$ out of $\mathcal{C}$ and decide for the codeword $\mathbf{c}^*$ which was most likely sent. Thus, the ML codeword $\mathbf{c}^*$ is the codeword which maximizes the so called conditional probability $p(\tilde{\mathbf{c}}$ received$|\mathbf{c}$ sent$)$:

$$\mathbf{c}^* = \arg\max_{\mathbf{c}\in\mathcal{C}} p(\tilde{\mathbf{c}} \text{ received}|\mathbf{c} \text{ sent}) \tag{2}$$

This problem is an optimization problem. In the following, we will present how decoding can be done with methods of integer optimization.

II. Optimization Methods for ML Decoding

Mathematical optimization is the task of finding an optimal solution for and the optimal value of an objective function under the condition that some constraints are satisfied. Objective function and side constraints typically model some real-world scenario, where the variables of the function are the abstraction of parameters like the per-day work time of different machines or the path on which a truck drives from location A to location B. Depending on the task, "optimal" may be either minimal (if the objective function represents some sort of cost or time) or maximal (e.g., if the function value stands for the profit gained). The constraints are used to model limitations of resources, working time, money, etc., or to avoid solutions that would be mathematically correct, but do not make sense in the real world—for example, a machine cannot work -2 hours per day, and one cannot build 1.72 facilities.

Linear programming is the best-studied discipline of mathematical optimization and it deals with optimization of a *linear* objective function which is constrained by linear (in)equalities. The domain of the variables is $\mathbb{R}^n$ for some $n \in \mathbb{N}$ and a *linear program (LP)* is then defined as

$$\begin{aligned} \min(\text{or } \max)\ w^T x &= \sum_{i=1}^{n} w_i \cdot x_i \\ \text{such that} \quad Ax &\leq b \\ x &\in \mathbb{R}^n \end{aligned}$$

where $w \in \mathbb{R}^n$ is the cost (benefit) vector, A the $m \times n$ constraint matrix, $b \in \mathbb{R}^m$ the right hand side vector of the constraints, and the variables are real-valued. Linear programming has become famous since George Dantzig in 1947 proposed an algorithm to solve LPs as the above one, called the Simplex algorithm. Despite the fact that the Simplex algorithm was later proven to have exponential worst-case complexity, it is still the algorithm most used in practice. For the far most problem instances, the Simplex method is very fast, and outperforms polynomial-time algorithms that were developed in the 1980's. Since its invention, linear programming has become a standard tool in business for a wide range of applications such as planning, scheduling, routing or facility location.

Yet, this is not the end of the story—for many applications, linear programs lack some important capabilities, such as the restriction to *integral* or *binary* decision variables.

From this requests, the field of *integer linear programming* evolved. An integer program (IP) is defined in the same way as a linear program, but additionally requires that some (or all) of the variables take values in $\mathbb{Z}^n$ instead of $\mathbb{R}^n$. The increased power obtained by this extension vastly enlarges the number of practical problems that can be modelled by IPs—at the cost, however, of efficiency, since it was shown that the general integer program is NP-hard to solve. Nevertheless, since this way of modelling is so powerful, a lot of research has been done to find and improve algorithms that tackle integer programs and make them solvable at least for moderate problem sizes or specific well-understood classes of programs.

How can integer linear programming techniques be used to decode binary linear codes and to add some surplus to the state-of-the-art knowledge in channel coding?

Jon Feldman was one of the first researchers who addressed this question intensively. To this end, he reformulated the ML problem (2) and interpreted it as an integer programming problem: Under the assumption of a *memoryless* channel, i.e. the error distributions are independent for subsequent bits, it can be calculated as

$$\begin{aligned} \mathbf{c}^* &= \arg\max_{\mathbf{c}\in\mathcal{C}} \left(\prod_{i=1}^{n} p(\tilde{\mathbf{c}}_\mathbf{i}|\mathbf{c}_\mathbf{i}) \right) \\ &= \arg\min_{\mathbf{c}\in\mathcal{C}} \left(\sum_{i:c_i=1} \ln\left(\frac{p(\tilde{\mathbf{c}}_\mathbf{i}|0)}{p(\tilde{\mathbf{c}}_\mathbf{i}|1)} \right) \right). \end{aligned}$$

The quotient

$$y_i = \ln\left(\frac{p(\tilde{\mathbf{c}}_\mathbf{i}|0)}{p(\tilde{\mathbf{c}}_\mathbf{i}|1)} \right) \tag{3}$$

is called the *log likelihood ratio* (LLR) of the i-th input bit. It only depends on the parameters of the channel, which are assumed to be known, and the signal $\tilde{\mathbf{c}}$ that was received. So, y_i is known to the decoder. From the above calculation it follows that $\mathbf{c}^* = \arg\min_{\mathbf{c}\in\mathcal{C}} (\sum_{i=1}^{n} y_i \mathbf{c}_\mathbf{i})$.

ML decoding can be achieved by minimizing this linear objective function over all codewords $\mathbf{c}$ that satisfy (1); in other words,

$$\begin{aligned} \min \sum_{i=1}^{n} y_i \mathbf{c}_\mathbf{i} \\ \text{such that} \quad H \cdot \mathbf{c} &\equiv 0 \bmod 2 \\ \mathbf{c} &\in \{0,1\}^n. \end{aligned}$$

Due to the modulo constraints, this is not exactly an integer programming model of the ML decoding problem. However, there exist several integer programming re-formulations of this optimization problem. For example, Jon Feldman introduced a formulation in which the number of both variables and constraints grows exponentially in the number of ones per

row [2]. Among others, this exponential growth led us to the development and investigation of the following improved integer programming model with less constraints and variables by the introduction of artificial variables $z \in \mathbb{Z}^m$ (one for each row of H)

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} y_i \mathbf{c_i} \\ \text{such that} \quad & H \cdot \mathbf{c} - 2z = 0 \\ & \mathbf{c} \in \{0,1\}^n,\ z \in \mathbb{Z}^m \end{aligned} \tag{4}$$

The j-th row of the constraint system reads $\sum_{i=1}^{n} H_{i,j} \cdot \mathbf{c_i} - 2z_j = 0$. Since z_j is required to be integral, this equation ensures that an even number of entries of c_i is set to 1. So, indeed this is a simple, general, and sparse integer programming problem for the ML decoding problem [3]. For any given code specified by a parity check matrix H, the solution to this problem is the maximum likelihood codeword.

It should be emphasized that this problem is difficult to solve (it is NP-hard in general). Therefore, the integrality requirements on the variables $\mathbf{c}$ and z are dropped in order to get the so-called LP-relaxation of this IP which can be considered an approximation of the original problem.

Our IP formulation proved to be beneficial in several ways.

First, in [4], we solved this IP formulation by a commercial all-purpose solver and compared its results with those of five other IP formulations existing in the literature. To sum it up, our formulation had the fewest number of variables as well as constraints and the solution time was the least.

Second, this formulation allows for an efficient procedure to approximately solve this problem (this is important if commercial solvers are not efficient enough or cannot be used due to some constraints). In [3], a separation algorithm was proposed utilizing the structural properties of this IP formulation. This algorithm works as follows: First, the LP solution is computed and checked for integrality. If it is integral, it is obviously also the optimal IP solution and the algorithm terminates. Otherwise, the IP-formulation is improved by adding additional linear constraints $\lambda^T \mathbf{c} \leq \lambda_0, (\lambda, \lambda_0) \in \Lambda$ where $\Lambda \subset \mathbb{R}^{n+1}$ is a set depending on some polyhedral properties of the code (which shall not be specified due to intended brevity of this article). However, these constraints ensure that if the LP obtained by adding this constraint is solved, then the previous LP solution is infeasible and the new LP solution is closer to the IP solution. Applying this idea iteratively yields an algorithm which approximates the IP solution. Consequently, the LP to be solved repeatedly is of the form

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} y_i \mathbf{c_i} \\ \text{such that} \quad & H \cdot \mathbf{c} - 2z = 0 \\ & \lambda^T \mathbf{c} \leq \lambda_0 \quad (\lambda, \lambda_0) \in \Lambda \\ & 0 \leq \mathbf{c_i} \leq 1, \quad i = 1, \ldots, n. \end{aligned}$$

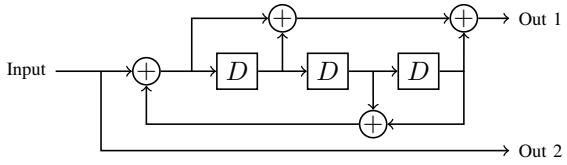

Fig. 6. A convolutional encoder with 3 memory registers.

The constraints $\lambda^T \mathbf{c} \leq \lambda_0, (\lambda, \lambda_0) \in \Lambda$, are derived from the current LP solution based on some combinatorial reasoning and the indicator variables z allow for a very fast generation of these constraints. Moreover, these constraints can be classified with respect to optimization theory: although they are derived in a combinatorial manner, they correspond to what is known as *Gomory cuts*. Additionally, it turned out that the formulation (4) offers another possibility which leads to an algorithm with enhanced performance: The representation of the code in terms of a parity check matrix H is not unique. In our separation algorithm, alternative parity check matrices of the code are generated during the execution as soon as the current one does not allow for further improvement of the solution.

To sum it up, this formulation enabled the development of an algorithm which

- is applicable to *any* binary linear block code in contrast to existing heuristics which are tailored to one specific type of code,
- yields a better error-correcting performance than other algorithms, and
- employs properties of binary linear codes to improve the algorithmic treatment.

Besides this general purpose formulation, we have also invented and investigated specialized approaches for an important subclass of codes, as described below.

III. Linear Programming for Trellis-Based Codes

A class of codes that have gained tremendeous interest since their invention in 1993 are the so-called *turbo codes*. A turbo code is built by the parallel concatenation of two convolutional codes that are separated by an interleaver. Convolutional codes have an inherent graph structure by their *trellis representation* which faciliates the processes both of encoding and decoding. This structure is partly inherited by the more complex turbo codes, which allowed us to invent a powerful integer programming formulation, as well as several approximation algorithms, for this special set of codes.

The encoder of a convolutional code is built by a number of delay shift registers (D) and several XOR (binary addition) gates, as shown in Figure 6. The registers comprise the memory of the encoder, and the (binary) content of the registers define its state. Thus, an encoder with k registers has 2^k possible states. By the XOR gates, the current state of the encoder, together with the current input bit, determines both the two current output bits at Out 1 and Out 2 and new state for the next step. This process can be visualized by the *trellis graph* of the code, where each vertex represents a state at a

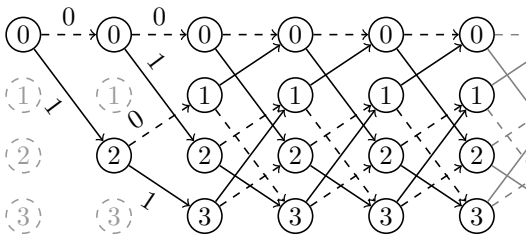

Fig. 7. Excerpt from a trellis graph with 4 states.

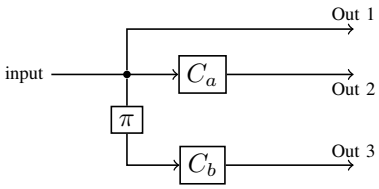

Fig. 8. Turbo encoder with two convolutional encoders C_a, C_b and interleaver π.

specific step, and there are two types of arcs for zero and one inputs, respectively, that model the transition of one state to another (see Figure 7 for a trellis with 4 states). One usually assumes that initially all registers contain a zero, therefore at the first step there is only one possible state. Arcs representing a zero input bit are drawn in a dashed style, where solid arcs correspond to a one at the input.

In a trellis, a codeword corresponds to a *path* from the initial state to the left to the end state at the right (which is not visible in the figure). It is possible to assign a cost attribute to each arc, based on the LLR values from (3), in a way that the ML codeword can be found by computing the path of minimum cost—a task for which very fast algorithms exist. Therefore, for a simple convolutional code, ML decoding can be accomplished efficiently.

The encoder of a turbo code uses two convolutional encoders. The first one processes the input bits as described above. A copy of the input is sent into the second encoder after passing an *interleaver* π. The interleaver permutes the bits in a pseudo-random way before they enter the second encoder. This way, the input (and thus also the output) is different for both decoders. By this additional complexity, the theoretical error-correcting performance of a turbo code is vastly increased compared to single convolutional encoding, as vital research over the past decades has revealed. The downside of this is that also the *decoding* complexity, the effort to compute the ML codeword for a given received signal, increases. Speaking in terms of trellis graphs, a turbo code can be modelled by two trellises which are synchronized by the constraint that in a given step, the path in the first trellis must use the same type of arc (zero or one) as the corresponding step in the second trellis which is determined by the interleaver function. The ML codeword is now represented by two paths (one for each trellis) of minimum cost that match this additional constraints defined by the interleaver. Unfortunatelly, it turns out that this problem is much harder than the single, unconstrained min-cost path in the case of a convolutional code. However, we can formulate the synchronized trellises as an integer linear program, where the interleaver constraints are of the form

$$\sum_{a \in S^i_{1,1}} x_a = \sum_{a \in S^{\pi(i)}_{2,1}} x_a. \tag{5}$$

Here, $S^i_{1,1}$ are the one-arcs in the first trellis for step i, and $S^{\pi(i)}_{2,1}$ are the one-arcs in the second trellis for the step to which the interleaver maps i.

We have run numerical simulations, showing that this formulation leads to greatly reduced computation time compared to the generic one given in (4). This is because only a rather small part of the problem definition, namely the interleaver constraints (5), separate the problem from the easy problem of unconstrained shortest path computation. *Lagrangian relaxation* can be used in such a situation of several "complicating constraints". We have adapted this technique to the case of turbo codes [5]. The idea of Lagrangian relaxation is to remove those constraints, but penalize their violation in the objective function. In the context of turbo coding, this means that we allow the two paths to use different types of arcs in the steps that are interconnected by the interleaver, but each of this violations increases the total cost of the path—the constraint given in (5) leads to the term

$$\eta_i \left(\sum_{a \in S^i_{1,1}} x_a - \sum_{a \in S^{\pi(i)}_{2,1}} x_a \right)$$

in the objective function which becomes larger the more the two sums differ.

A central result of Lagrangian relaxation is that, for an appropriate choice of penalty factors η_i, the optimal paths with respect to the Lagrangian relaxation will converge to the LP relaxation of the problem, where all additional constraints are satisfied. However, in general no integral solution is obtained.

One of our approaches to finding integral solutions is based on the following idea: Given the shortest path on one of the trellises, it is rather unlikely that the corresponding path on the other trellis (determined by the interleaver) is also the shortest path for that trellis. However, the optimal global solution corresponds to the k-th shortest path on one of the trellises, and under certain assumtions k will be relatively small k. We have adapted an efficient algorithm to compute the k shortest paths on a graph to the trellis scenario, and showed that, for low noise and not too large trellises, this can be an efficient approach—especially in conjunction with the modified objective function by the Lagrangian relaxation.

IV. Results, Impact, & Outlook

Remember that in practical communication systems the use of an optimal decoder is very often not feasible, because of its large complexity. So when implementing decoders on a chip suboptimal algorithms are used. They are much less complex than the optimal decoder, but also perform worse. The algorithms are often iterative heuristics, that usually run

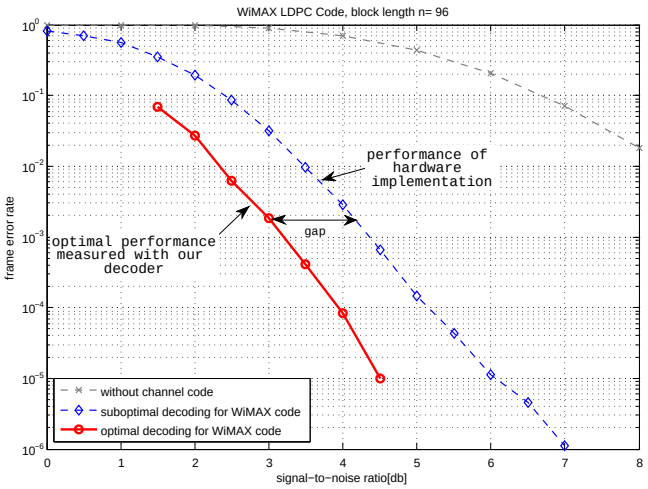

Fig. 9. Comparison of suboptimal and optimal decoding

a fixed number of iterations or abort earlier according to some stopping criterion. For example the decoding of LDPC codes in practical applications is usually done with a special iterative algorithm called belief-propagation, that is also used in artificial intelligence [6].

Finding decoding algorithms with a good tradeoff between hardware complexity and communications performance is an important research area for hardware designers. To judge an algorithm's performance it can be compared to the optimal decoding. This comparison also helps to estimate if further research on new algorithms is promising or not. However the problem is that for most linear block codes used in practice the optimal decoding performance is unknown.

Our aim is to provide ML simulation results for practical channel codes with larger blocklengths. Furthermore with mathematical techniques we like to get further insight in channel codes and their decoding procedure.

In chapter I we presented the optimal decoding problem as integer optimization problem. With help of the presented techniques we built an optimal decoder, that is able to perform ML decoding efficiently for larger block lengths. With this decoder we measured the optimal performance of some short channel codes used in the recent communication standard WiMAX [7], which were previously unknown.

Figure 9 shows a LDPC code taken from WiMAX with a block lengths of 96 bits. The suboptimal performance as it may be implemented on a chip is shown. Additionally a second curve is shown, which was simulated with the use of our integer optimization decoder and shows the optimal performance.

One can see that there is a gap of about 1 db between the algorithm used in hardware and what is theoretically achievable. This gap is quite large, if you consider that sometimes a few tenths of a db are a good sales argument.

The work on ML decoding as an optimization problem did not only lead to a better understanding of the decoding performance of the suboptimal algorithms and new simulation results. The study of different formulations also led to a new stopping criterion for a iterative decoding algorithm. Stopping criterions abort the iterative process early, if sufficient iterations were made. Here we used a cost function to indicate, when the iterations can be stopped. This leads to a faster computation and a more power efficient decoding algorithm in hardware.

The study of linear block codes gives rise to a lot of new types of mathematical problems and structures that have not been considered before, as well as interesting links to different mathematical fields (like integer programming, graph and network theory, finite algebra, matroid theory). If we use integer programming for ML decoding, we are in the situation that, for a given code, we solve the same program over and over again, where only the cost vector varies. So far, only little attention has been given to this special case, which might allow for more efficient ways to solve the problem by re-using information that was collected in previous problem instances. As another example, by studying certain types of cyclic codes we encountered a new type of network flow problem, where the supply or demand of vertices is variable and thus a part of the problem itself, but restricted to be a multiple of 2. Also, the application of coding theory might arouse interest in constraints of the form $Ax \equiv b \pmod{q}$ in integer programs, which have not yet been studied extensively.

The recently developed 3-dimensional turbo codes, which are an extension of turbo codes that use a third convolutional encoder in conjunction with a second interleaver, show that the development of good Lagrangian relaxation techniques for trellis-like graphs remains an important task for the area of convolutional coding.

ACKNOWLEDGEMENT

We thank Horst W. Hamacher, Frank Kienle, Stefan Ruzika, and Norbert Wehn for their constructive comments and suggestions. We gratefully acknowledge financial support by the Center of Mathematical and Computational Modelling of the University of Kaiserslautern.

REFERENCES

[1] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[2] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[3] A. Tanatmis, S. Ruzika, H. Hamacher, M. Punekar, F. Kienle, and N. Wehn, "A separation algorithm for improved LP-decoding of linear block codes," *Information Theory, IEEE Transactions on*, vol. 56, no. 7, pp. 3277–3289, 2010.

[4] A. Tanatmis, S. Ruzika, M. Punekar, and F. Kienle, "Numerical Comparison of IP Formulations as ML Decoders," in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.

[5] A. Tanatmis, S. Ruzika, and F. Kienle, "A lagrangian relaxation based decoding algorithm for LTE turbo codes," in *Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on*. IEEE, 2010, pp. 369–373.

[6] T. Richardson and R. Urbanke, "The Renaissance of Gallager's Low-Density Pariy-Check Codes," *IEEE Communications Magazine*, vol. 41, pp. 126–131, Aug. 2003.

[7] *IEEE 802.16*, Local and metropolitan area networks; Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems; Amendment 2:Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands.