# Optimization of Turbo Codes by Differential Evolution and Genetic Algorithms

Pavel Krömer, Václav Snášel, Jan Platoš and Ajith Abraham

*FEECS, Dept. of Computer Science*
*VŠB – Technical University of Ostrava*
*17. listopadu 15, CZ 708 33 Ostrava–Poruba, Czech Republic*
*{ pavel.kromer.fei, vaclav.snasel, jan.platos.fei} @ vsb.cz, ajith.abraham @ ieee.org*

## Abstract

*Since their appearance in 1993, first approaching the Shannon limit, turbo codes gave a new direction in the channel encoding field, especially since they have been adopted for multiple telecommunication norms. To obtain good performance, it is necessary to design a robust turbo code interleaver. This paper proposes a a differential evolution approach to find above average turbo code interleavers. Performance is compared with the conventional genetic algorithm approach and the empirical results illustrate that DE performs well.*

## 1. Introduction

The introduction of the turbo principle allowed close approach to the Shannon limit – a theoretical boundary describing the maximum capacity of a noisy communication channel. The invention of the turbo codes and its superior performance in practical applications also initiated a renaissance of channel coding research [1, 2]. The exceptional performance of the turbo codes can be further improved by finding right settings for a particular system. The structure of the interleaver, performing a permutation of input bits, is one important property of any turbo code system. In this research, we optimize the turbo code interleaver by genetic algorithms and differential evolution.

## 2. Turbo codes

The turbo codes were introduced by Berrou, Glavieux and Thitimajshima in 1993 [*2*] and they have become a hot research topic since then. Prior to the turbo codes, $3dB$ or more separated the spectral efficiency of real world channel encoding systems from the theoretical maximum described by Shannon theorem [*2*].

The turbo codes are an implementation of parallel concatenation of two circular recursive systematic convolutional (CRSC) codes based on a pseudo-random permutation (the interleaver). The general scheme of a classic turbo encoder is shown in Figure 1.

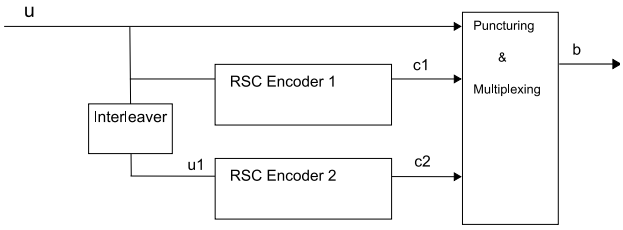


**Figure 1.** A general scheme of turbo encoder

The encoder processes a $N$-bit long information frame $u$. The input frame is interleaved by the $N$-bit interleaver to form permuted frame $u_1$. Original input frame is encoded by the encoder RSC1 and interleaved frame is encoded by RSC2. Hereafter, the two encoded frames $c_1$ and $c_2$ are merged together and with the original input sequence $u$ according to some puncturing and multiplexing scheme. The rate of the code is defined as $r = \frac{k}{n} = \frac{InputSymbols}{OutputSymbols}$.

Previous studies have illustrated that a random block interleaver (random permutation of the input frame) can be in certain cases (e.g. for $BER > 10^{-5}$) more efficient than other channel encoding schemes [3]. In this paper, an optimized turbo code block interleaver is compared with a random block interleaver by means of BER to evaluate its efficiency.

978-0-7695-3745-0/09 $25.00 © 2009 IEEE
DOI 10.1109/HIS.2009.289

The increase of the interleaver size gives better performance and better interleaving gain while worsening latency. Equation (1) illustrates the influence on the latency:

$$t_d = \frac{K_f}{R_b} N_i \quad (1)$$

In (1), $R_b$ is the code bit rate, $K_f$ stands for the frame size and $N_i$ is the number of the decoding stages.

Interleaver matrix sizes vary from tens to ten-thousands of bits. When optimizing, it is highly inefficient, if not impossible, to test all the possible input vectors $2^N$ with all the possible $N!$ interleaver matrices, requiring $2^N N!$ tests in total. Therefore, advanced interleaver optimization methods are under investigation.

### 2.1. Interleaver Evaluation

As noted earlier, the performance of a turbo code can be evaluated by the means of bit error rate, i.e. the ratio of the number of incorrectly decoded bits to the number of all bits transmitted during some period. Unfortunately, it is rather hard to compute the BER for a turbo code and the simulations can be for larger interleavers very inaccurate. The error floor of a *C(n, k)* code can be analytically estimated:

$$BER \cong \frac{w_{free}}{2k} erfc\left(\sqrt{d_{free} \frac{k}{n} \frac{E_b}{N_0}}\right) \quad (2)$$

To estimate BER, the following code properties must be known [4]:

- $d_{free}$- the free distance, i.e. the minimum number of different bits in any pair of codewords
- $N_{free}$ - the free distance multiplicity, i.e. the number of input frames generating codewords with $d_{free}$
- $w_{free}$- the information bit multiplicity, i.e. the sum of the Hamming weights of the input frames generating the codewords with $d_{free}$

There are several algorithms for free distance evaluation. Garello et al. [4] presented an algorithm designed to effectively compute free distances of large interleavers with unconstrained input weight based on constrained subcodes.

This work introduces interleaver optimization driven by algebraical estimation of maximum $d_{free}$ evaluated using analytical approach.

## 3. Genetic Algorithms

Genetic algorithms are probably the most popular and wide spread member of the class of evolutionary algorithms (EA). EAs form a group of iterative stochastic search and optimization methods based on mimicking successful optimization strategies observed in nature [5, 6, 7, 8]. The essence of EAs lies in their emulation of Darwinian evolution, utilizing the concepts of Mendelian inheritance for use in computer science [6]. Together with fuzzy sets, neural networks, and fractals, evolutionary algorithms are among the fundamental members of the class of soft computing methods.

EAs operate with a population of artificial individuals (also referred to as items or chromosomes) encoding possible problem solutions. Encoded individuals are evaluated using a carefully selected objective function which assigns a fitness value to each individual. The fitness value represents the quality (ranking) of each individual as a solution to a given problem. Competing individuals explore the problem domain towards an optimal solution [8].

Genetic algorithms (GA) introduced by John Holland and extended by David Goldberg, are a widely applied and highly successful EA variant. Evolutionary principles are in GA implemented via iterative application of so called genetic operators: the mutation, the crossover and the selection. Mutation and selection can be found in more evolutionary techniques while crossover is significant for genetic algorithms. Many variants of the standard generational GA have been proposed. The main differences lay mostly in particular selection, crossover, mutation, and replacement strategies [8].

Genetic algorithms have been successfully used to solve non-trivial multimodal optimization problems. They inherit the robustness of emulated natural optimization processes and excel in browsing huge, potentially noisy problem domains. Their clear principles, ease of interpretation, intuitive and reusable practical use and significant results made genetic algorithms the method of choice for industrial applications, while carefully elaborated theoretical foundations attract the attention of academics.

## 4. Genetic Algorithms for Interleaver Optimization

The optimization of a turbo code interleaver is a non-trivial combinatorial optimization task. Genetic algorithms have already been considered for interleaver matrix optimization. Durand et al. [9] used customized GA to optimize an interleaver of the size

105, comparing their results to previous interleaver design techniques. The genetic algorithm that was used was fully based on mutation and the crossover operator was omitted. The fitness criterion for every interleaver was maximum free distance.

Rekh et al. [10] presented another GA variant for interleaver optimization, introducing 2-point crossover to the interleaver evolution process. Nevertheless, the crossover impact was influenced by the necessary correction of errors caused by crossover applications. The fitness criterion was BER and the size of the optimized interleaver $N$ was 50.

## 4.1. Interleaver GA Design

An interleaver with the dimension $N$ performs a hardware permutation of $N$ input bits and it can be seen as a general permutation of $N$ symbols $\sigma_N = (i_1, i_2, \dots, i_N)$, where $i_k \epsilon [1, N]$ and $i_n \neq i_m$ for all $m \neq n \epsilon [1, N]$. The application of $\sigma_N$ on a sample input vector $I_N$ for $N = 5$ is illustrated in (3).

$$\begin{aligned} I_5 &= (0, 1, 0, 1, 1, 1) \\ \sigma_5 &= (5, 3, 4, 1, 2) \\ O_5 &= \sigma_5(I_5) = (1, 0, 1, 0, 1) \end{aligned} \tag{3}$$

Intuitively, $\sigma_N$ can be used as a permutation encoding for genetic algorithms, as utilized e.g. in [10]. Durand et al. [5] did not explicitly specify used interleaver encoding, although we can conclude that they used a similar interleaver representation.

In our GA implementation for turbo code interleaver optimization two types of selection techniques were used. In order to speed up the convergence of the algorithm, a semi-elitary hybrid selection scheme choosing one parent by elitary manners and the second by proportional manners of roulette wheel selection. Mutation was implemented by swapping the positions of two coordinates in $\sigma_N$. On the contrary, traditional crossover operators (except uniform crossover) would corrupt the structure of intuitively encoded permutation $\sigma_N$ and hence they cannot be used without some post-processing used for chromosome fixing. This is a remarkable fact, since crossover is referred to as the primary operator for GA [7].

## 4.2. The Role of Crossover in GA

The crossover operator is the main operator of genetic algorithms that distinguishes it from other stochastic search methods [7]. Its role in the GA process has been intensively investigated and its omission or traversing is expected to affect the efficiency of a GA solution significantly.

A crossover operator is primarily a creative force in the evolutionary search process. It is supposed to propagate building blocks (low-order, low defining-length schemata with above-average fitness) from one generation to another and create new (higher-order) building blocks by combining low order building blocks. It is intended to introduce large changes to the population with low disruption of building blocks [11]. In contrast, mutation is expected to insert new material to the population by random perturbation of chromosome structure. In this way, however, new building blocks can be created or old ones disrupted [11].

An experimental study on crossover and mutation in relation to the frequency and lifecycle of building blocks in a chromosome population showed that the ability of mutation and one-point crossover to create new building blocks is almost the same. However, crossover is irreplaceable for spreading newly found building blocks among the population (which can lead to loss of diversity in the population) [11].

A mutation-only genetic algorithm avoids the use of a crossover operator. This, as already mentioned, can be seen as a significant weakening of the algorithm.

Another strategy available for problems involving permutation evolution is the random keys encoding [12]. With random keys encoding, the permutation is represented as a string of real numbers (random keys), whose position after ordering corresponds to the permutation index. Random keys encoded chromosomes and the crossover of randomly encoded permutations are illustrated in Figure 2. When there are two identical random keys in the population, randomly encoded permutations can be corrupted by crossover too. However, the occurrence of two equal random keys can be considered as a very rare situation.

There are two other more significant drawbacks of the RK encoding. First, it is rather computationally expensive. It requires sorting of the real array every time a chromosome is modified and needs to be evaluated. Second, genetic algorithms were not designed to deal with real-valued chromosomes, even though there are methods on how to process real-valued chromosomes by GA.

A chromosome with Random Key Encoding:

| Random key: | 0.2 | 0.1 | 0.3 | 0.5 |
|---|---|---|---|---|
| Allele: | 2 | 1 | 3 | 4 |

One Point Crossover of two RKE chromosomes:

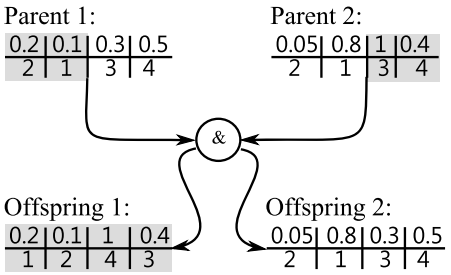


**Figure 2.** Random key encoding examples

## 5. Differential Evolution

Differential evolution (DE) is a reliable, versatile and easy to use stochastic evolutionary optimization algorithm [13]. DE is a population-based optimizer that evolves real encoded vectors representing the solutions to given problem. The real-valued nature of population vectors differentiates the DE notably from GAs that were originally designed to evolve solution encoded into binary or finite discrete alphabets.

The DE starts with an initial population of N real-valued vectors. The vectors are initialized with real values either randomly or so, that they are evenly spread over the problem domain. The latter initialization leads to better results of the optimization process [13].

During the optimization, DE generates new vectors that are perturbations of existing population vectors. The algorithm perturbs vectors with the scaled difference of two (or more) randomly selected population vectors and adds the scaled random vector difference to a third randomly selected population vector to produce so called trial vector. The trial vector competes with a member of the current population with the same index. If the trial vector represents a better solution than the population vector, it takes its place in the population [13].

Differential evolution is parameterized by two parameters [13]. Scale factor $F \in (0,1+)$ controls the rate at which the population evolves and the crossover probability $C \in [0,1]$ determines the ratio of bits that are transferred to the trial vector from its opponent. The number of vectors in the population is also an important parameter of the population. The outline of classical DE algorithm is shown in Figure 3.

I. Initialize the population $P$ consisting of $M$ vectors
II. Evaluate an objective function ranking the vectors in the population
III. Create new population:
For $i \in \{1 \,..\, M\}$:
   a. Create a trial vector $v_t^i = v_r^1 + F \cdot (v_r^2 - v_r^3)$, where $F \in [0,1]$ is a parameter and $v_r^1, v_r^2, v_r^3$ are three random vectors from the population $P$. This step is in DE called mutation.
   b. Validate the range of coordinates of $v_t^i$. Optionally adjust coordinates of $v_t^i$ so, that $v_t^i$ is valid solution to given problem.
   c. Perform uniform crossover. Select randomly one point (coordinate) $l$ in $v_t^i$. With probability $1 - C$ let $v_t^i[m] = v^i[m]$ for each $m \in \{1, \ldots, N\}$ such that $m \neq l$
   d. Evaluate the trial vector. If the trial vector $v_t^i$ represent a better solution than population vector $v^i$, replace $v^i$ in $P$ by $v_t^i$
IV. Check termination criteria; if not satisfied go back to III.

**Figure 3.** A summary of DE algorithm

There are more variants of differential evolution. They differ mostly by the way new vectors are generated. We used the classical differential evolution algorithm, also referred to as DE/rand/1/bin [13].

Differential evolution is prospective method for turbo code interleaver optimization since it operates on real valued vectors and a permutation represented by RK encoding is indeed a real vector. Moreover, differential evolution has been shown to outperform genetic algorithms in some problem domains [13].

## 6. Interleaver Optimization Experiments

A simulation framework built upon the IT++ library[1] was used to experimentally evaluate the proposed interleaver generation method. IT++ is a robust and efficient C++ library of mathematical and telecommunication algorithms and objects. It provides high native code performance and excellent abstraction of a well-defined object-oriented framework. We have experimented with 64-bit, 128-bit and 256-bit interleavers. The settings for optimization experiments were:

- Random keys encoding of chromosomes
- 1000 generations (GA), 3000 and 5000 evaluations (DE)
- Probability of crossover 0.8 and probability of mutation 0.2 for GA

- C equal to 0.9 and F equal to 0.9 for DE
- Population of 20 vectors for DE and population of 40 chromosomes for GA

The fitness function used in GA and DE for turbo code interleaver optimization was based on analytical $d_{free}$ estimation as defined in [4]. The goal of the fitness function was to maximize $d_{free}$, and minimize $N_{free}$ and $w_{free}$:

$$f_{fitness} = A \cdot d_{free} - B \cdot N_{free} - C \cdot w_{free} \qquad (4)$$

where the coefficients $A$, $B$ and $C$ were fixed to 100, 10 and 1 respectively. All experiments were executed several times to overcome the stochastic nature of the algorithms.

### 6.1. Optimization Results

The results of experimental interleaver optimization are summarized in Tables 1 and 2. Table 1 outlines the average obtained interleaver free distance for 64, 128 and 256-bit interleavers. The first column shows average free distance of a random interleaver, second column shows average $d_{free}$ of an inteleaver optimized by GA and third and fourth column show average $d_{free}$ of an interleaver optimized by DE after 3000 and 5000 evaluations respectively.

| N | avg random | avg GA | avg DE (3000) | avg DE (5000) |
|---|---|---|---|---|
| 64 | 13,50 | 15,70 | 15,90 | **16,25** |
| 128 | 14,25 | 17,90 | 17,60 | **18,25** |
| 256 | 15,75 | **20,20** | 18,20 | 18,25 |

**Table 1.** Comparison of average free distances of random interleaver, interleaver evolved by GA and interleavers evolved by DE.

Table 2 represents maximum values of $d_{free}$ for investigated interleavers.

| N | max random | max GA | max DE (3000) | max DE (5000) |
|---|---|---|---|---|
| 64 | 13 | **17** | 16 | **17** |
| 128 | 14 | **19** | 18 | **19** |
| 256 | 17 | **21** | 19 | 20 |

**Table 2.** Comparison of average free distances of random interleaver, interleaver evolved by GA and interleavers evolved by DE

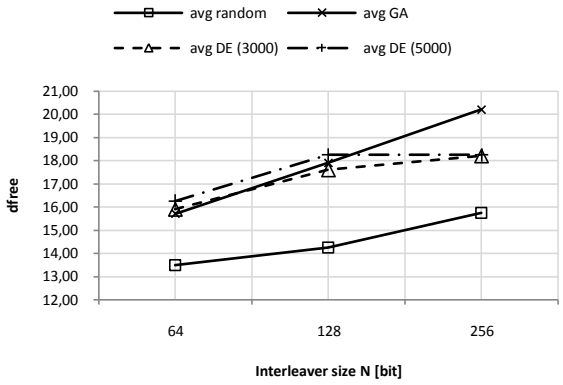


**Figure 4.** Visual comparison of average $d_{free}$

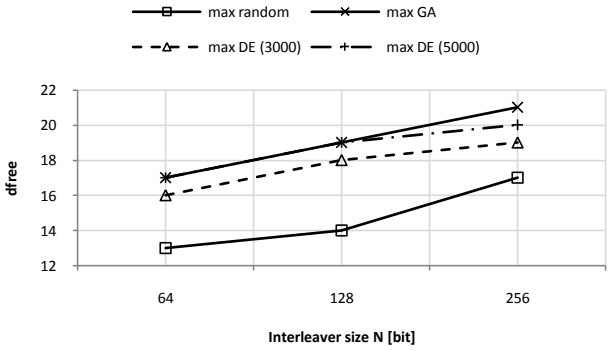


**Figure 5.** Visual comparison of maximum $d_{free}$

The experiment results illustrate that both algorithms are powerful interleaver optimizers. Differential evolution provided better average free distance for 64 and 128-bit interleavers. For 256-bit interleaver GA delivered the best empirical performance.

The results of DE were observed after 3000 and 5000 evaluations of fitness function. 5000 DE evaluations required approximately the same time as GA with 1000 generations.

Both DE and GA obtained the same maximum $d_{free}$ values for 64 and 128-bit interleavers. In case of 256-bit interleaver delivered GA better maximum $d_{free}$.

## 7. Conclusions

This paper discussed the problem of efficient turbo code interleaver optimization by genetic algorithms and differential evolution. The turbo code interleavers in the form of permutations were encoded by random keys encoding method as vectors of real numbers. An analytical estimation of free distance was used as a basis of fitness function for both, GA and DE. The results suggest that both GA and DE can provide good

results. Both algorithms reached the same improved maximum $d_{free}$ for 64 and 128-bit interleavers. DE delivered for 64 and 128-bit interleavers better average $d_{free}$ across multiple experimental runs. Unlike for smaller interleavers, GA outperformed DE for 256-bit interleaver.

In our future work, we aim to investigate the performance of presented optimizers on more interleavers of different sizes to determine which algorithm performs better. Moreover, other DE variants can be also evaluated.

## References


[1] A. Burr, "Turbo-codes: the ultimate error control codes?," in *Electronics & Communication Engineering Journal*, vol. 13, pp. 155–165, IEEE, August 2001. ISSN: 0954-0695.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo codes," in *Proc. Int. Conf. on Commun.*, pp. 1064–1070, 1993.

[3] J. Hokfelt and T. Maseng, "Methodical interleaver design for turbo codes," *International Symposium on Turbo Codes*, 1997.

[4] R. Garello, F. Chiaraluce, P. Pierleoni, M. Scaloni, and S. Benedetto, "On error floor and free distance of turbo codes," in *IEEE International Conference on Communications (ICC 2001)*, vol. 1, pp. 45–49, 2001. ISBN 0-7803-7097-1.

[5] M. Dianati, I. Song, and M. Treiber, "An introduction to genetic algorithms and evolution strategies," technical report, University of Waterloo, Ontario, N2L 3G1, Canada, July 2002.

[6] U. Bodenhofer, "*Genetic Algorithms: Theory and Applications*," lecture notes, Fuzzy Logic Laboratorium Linz-Hagenberg, Winter 2003/2004.

[7] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[8] G. Jones, "Genetic and evolutionary algorithms," in *Encyclopedia of Computational Chemistry* (P. von Rague, ed.), John Wiley and Sons, 1998.

[9] N. Durand, J. Alliot, and B. Bartolomé, "Turbo codes optimization using genetic algorithms," in *Proceedings of the Congress on Evolutionary Computation* (P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, eds.), vol. 2, (Mayflower Hotel, Washington D.C., USA), pp. 816–822, IEEE Press, 6-9 1999.

[10] S. Rekh, S. Rani, W. Hordijk, P. Gift, and Shanmugam, "Design of an interleaver for turbo codes using genetic algorithms," *The International Journal of Artificial Intelligence and Machine Learning*, vol. 6, pp. 1–5, 2006.

[11] A. S. Wu, R. K. Lindsay, and R. Riolo, "Empirical observations on the roles of crossover and mutation," in *Proc. of the Seventh Int. Conf. on Genetic Algorithms* (T. Bäck, ed.), (San Francisco, CA), pp. 362–369, Morgan Kaufmann, 1997.

[12] L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006.

[13] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution A Practical Approach to Global Optimization*. Natural Computing Series, Berlin, Germany: Springer-Verlag, 2005.