International Journal For Technological Research In Engineering
Volume 2, Issue 1, September-2014 *ISSN (Online): 2347 - 4718*

INTERLEAVER OPTIMIZATION BY PARTICLE SWARM ALGORITHM

K.Prabhakar[1], Ch.Ravi Kumar[2], Dr.K.Padmaraju[3]
[1]PG student, [2]Asst. Prof, Department of E.C.E,
Sir C. R. Reddy College of Engineering, Eluru, India.
[3]Principal, JNTU Kakinada,
Kakinada, India.

Abstract: Turbo coding is an influential method of channel coding in telecommunications. 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. The exceptional performance of the turbo codes can be further improved by finding right settings for a particular system. Among others, the structure of the interleaver performing a permutation of input bits is one important property of any turbo code system. In this paper, we present genetic algorithms and particle swarm algorithm as two promising optimization methods to find a well performing turbo code interleaver.
Index terms: dfree, BER, interleaver, GA and PSO.

I. INTRODUCTION

The turbo codes were introduced by Berrou, Glavieux and Thitimajshima in 1993 [1] and they have become a hot topic soon after their introduction. 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 [1].Turbo coding brought to the world of channel encoding one important principle: the feedback concept, exploited heavily in electronics, to be utilized in decoding of concatenated codes and it was indeed the iterative decoding (the actual turbo principle) that helped the turbo codes to achieve its impressive near-optimum performance [1]. The performance of the turbo codes depends on two principal parameters; the first is the code spectrum and the second is the decorrelation between the external information at the same number of iterations. The optimization process can be used for of the matrix stature the amelioration of performance and the dimilution with safe performance. The latter is very interesting for multimedia real-time satellite transmission systems because the interleaving matrix causes a considerable dimilution of the codec complexity and delay. The original turbo coder [1] [2] was designed as a parallel concatenation of two circular recursive systematic convolutional codes. The key component that differentiates turbo codes from other concatenated codes is the interleaver, which permutes the input frame before it is processed by the second encoder. The interleaver acts as a pseudorandom block scrambler defined by a permutation with no repetitions [2]. One of the key principles behind the performance of turbo coding is the fact that each of the encoders typically produces a high-weight code word for most inputs, but it produces a low-weight (i.e. more error prone) code word for only few inputs. The interleaver makes it more unlikely that both encoders will output a bad code word for the same input, increasing the probability that the decoder will be able to extract the correct information. An interleaver is implemented by an interleaver matrix. Interleaver of the dimension N performs a hardware permutation of N input bits but it can be modelled as a general permutation of N symbols. Interleaver sizes vary from tens to tens of thousands. When searching for an optimal interleaver, it is computationally infeasible to test all possible interleaver matrices (N!) and all possible input vectors (2N) to find a globally best interleaver. Therefore, advanced interleaver optimization methods are sought. A search for an permutation of N symbols is a typical combinatorial optimization problem.

A. Interleaver evaluation

Turbo code performance can be evaluated by means of bit error rate (BER), the ratio of incorrectly decoded bits to the number of all bits of information transmitted during some period. It is hard to compute the BER for a turbo code and the simulations can be inaccurate for longer interleavers. If simulations are performed, particular model of the noisy communication channel must be used. The additive white Gaussian noise (AWGN) channel is popular model for telecommunication research. It is a good model for satellite and deep space communication links [3].Rayleigh fading channel is a reasonable model for tropospheric and ionospheric signal propagation as well as the effect of heavily built-up urban environments on radio signals [3]. An alternative to simulations represents analytical estimation of code error floor. The error floor of a C (n, k) code can be analytically estimated.

$$\text{BER} \approx \frac{wfree}{2k} \operatorname{erfc}\left(\sqrt{\text{dfree}\frac{k}{n}\frac{Eb}{N0}}\right) \quad (1)$$

where erfc is complementary error function. To estimate BER, the following code properties must be known [4]:

- dfree - the free distance, i.e. the minimum number of different bits in any pair of codewords
- Nfree-the free distance multiplicity, i.e. the number of input frames generating codewords with dfree
- wfree - the information bit multiplicity, i.e., the sum of the Hamming weights of the input frames generating the codewords with dfree

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.

II. GENETIC ALGORITHMS

Genetic algorithms (GA) are a well known population based metaheuristic soft optimization method [5]. Gas solves complex optimization problems by programmatic evolution of an encoded population of candidate problem solutions. The solutions are ranked using a problem specific fitness function. The artificial evolution is implemented by iterative application of genetic operators and leads to discovery of above average solutions. The basic workflow of the standard generational GA is shown in Algorithm 1.

1 Define objective (fitness) function and problem encoding;
2 Encode initial population P of possible solutions as fixed length strings;
3 Evaluate chromosomes in initial population using objective function;
4 while Termination criteria not satisfied do
5 Apply selection operator to select parent chromosomes for reproduction:
sel(Pi) → parent1, sel(Pi) → parent2;
6 Apply cross-over operator on parents with respect to cross over probability to produce new chromosomes: cross (pC, parent1, parent2) → {of f spring1, of f spring2};
7 Apply mutation operator on off spring chromosomes with respect to mutation probability: mut(pM, off spring1) → off spring1, mut(pM, off spring2) → off spring2;
8 Create new population from current population and offspring chromosomes: migrate (offspring1, off spring2,Pi) → Pi+1;
9 end Algorithm 1: A summary of genetic algorithm

The algorithm is shown in the flow chart below in Fig.1.

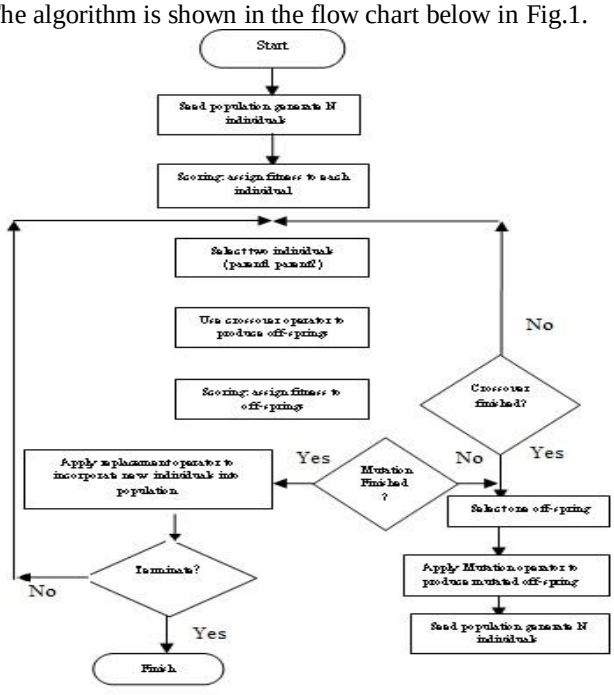

Fig.1: Flow chart of GA

Problem encoding is an important part of genetic search. It translates candidate solutions from the problem domain (phenotype) to the encoded search space (genotype) of the algorithm and defines the internal representation of the problem instances used during the optimization process. The representation specifies chromosomes data structure and the decoding function [6]. The data structure defines the actual search space, its size and shape. The choice of encoding also affects the set and implementation of applicable evolutionary operators. Encoding and evolutionary operators are closely connected because the operators operate on the data structure defined by the encoding [7]. Certain encodings, such as direct encoding of permutation, disable the application of crossover operator. Other encodings require reimplementation of commonly used genetic operators. In general, it is not easy to find suitable encoding that will allow the use of fully featured genetic algorithms for interleaver evolution or combinatorial optimization problems in general. For instance, the loss of crossover might be considered as significant weakening of the algorithm. Crossover operator is the main operator of the genetic algorithms distinguishing it from other population based stochastic search methods [5]. Its role in the GA process has been intensively investigated and its avoidance is expected to affect the efficiency of a GA solution negatively. 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 to the population large changes with small disruption of building blocks [8]. In contrast, mutation is expected to insert new material to the population by random perturbation of chromosome structure. By this, however, can be new building blocks created or old disrupted [8].

III. PSO ALGORITHM

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart in 1995. PSO is similar to Genetic Algorithm (GA) in that the system is initialized with a population of random solutions. It is unlike a GA, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called particles, are then "flown" through the problem space. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (This fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the global version of the particle swarm optimizer the overall best value, and its location, obtained so far by any particle in the population. This location is called gbest [11]. The particle swarm optimization concept consists of, at each time step, changing the velocity (accelerating) each particle toward its pbest and gbest locations (global version of PSO). Acceleration is weighted by a random numbers being generated for

acceleration toward pbest and gbest locations. This is also a local version of PSO in which, in addition to pbest, each particle keeps track of the best solution, called lbest, attained within a local topological neighbourhood of particles. The (original) process for implementing the global version of PSO is as follows:

1 Initialize a population (array) of particles with random positions and velocities on d dimensions in the problem space.

2 For each particle, evaluate the desired optimization fitness function in d variables.

3 Compare particle's fitness evaluation with particle's pbest. If current value is better than pbest, then set pbest value equal to the current value and the pbest location equal to the current location in d-dimensional space.

4 Compare fitness evaluations with the population's overall previous best. If current value is better than gbest, then reset gbest to the current particle's array index and value.

5 Change the velocity and position of the particle according to equations (1) and (2), respectively:

vid = vid + c1 * rand() * (pid - xid) + c2 * Rand() * (pgd - xid) (1)

xid = xid + vid (2)

6 Loop to step 2 until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations (generations).

Algorithm 2: A summary of particle swarm optimization.

The algorithm is shown in the flow chart below in Fig.2.

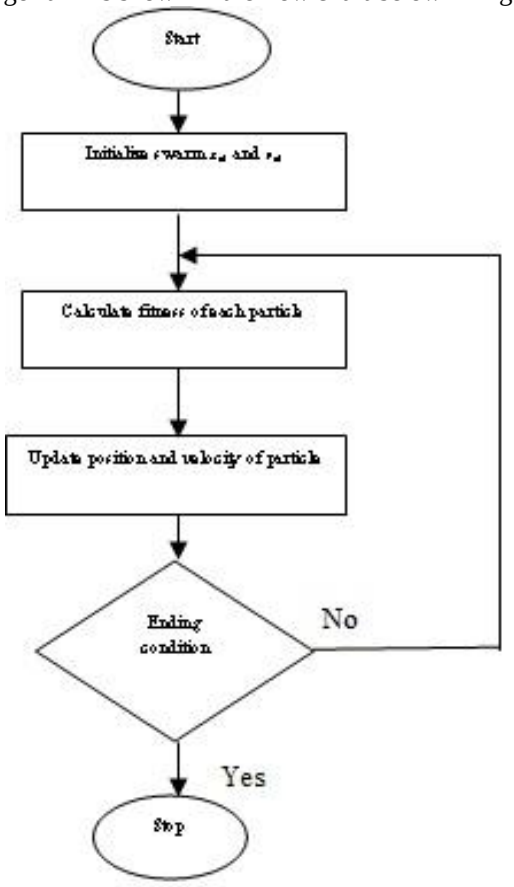

Fig.2: Flow chart of PSO

Particles velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of the accelerations would cause the velocity on that dimension to exceed Vmax, which is a parameter specified by the user, then the velocity on that dimension is limited on Vmax. Vmax is therefore an important parameter. It determines the resolution, or fineness, with which regions between the present position and the target (best so far) position are searched. If Vmax is too high, particles might fly past good solutions. If Vmax is too small, on the other hand, particles may not explore sufficiently beyond locally good regions. In fact, they could become trapped in local optima, unable to move far enough to reach a better position in the problem space [12]. The acceleration constants c1 and c2 in equation (1) represent the weighting of the stochastic acceleration terms that pull each particle toward pbest and gbest positions. Thus, adjustment of these constants changes the amount of "tension" in the system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement toward, or past, target regions. Early experience with particle swarm optimization (trial an error, mostly) led us to set the acceleration constants c1 and c2 each equal to 2.0 for almost all applications. Vmax was thus the only parameter we routinely adjusted, and we often set it about 10-20% of the dynamic range of the variable on each dimension. Based, among other things, on findings from social simulations, it was decided to design a "local" version of the particle swarm. In this version, particles have information only of their own and their neighbour's bests, rather than that of the entire group. Instead of moving toward a kind of stochastic average of pbest and gbest (The best location of the entire group), particles move toward points defined by the pbest and "lbest," which is the index of the particle with the best evaluation in the particle's neighbourhood. If the neighbourhood size is defined as two, for instance, particle (i) compares its fitness value with particle (i-1) and particle (i+1). Neighbours are defined as topological neighbours; neighbours and neighbourhoods do not change during run. For the neighbourhood version, the only change in process is defined in the six steps before is the substitution of pid, the location of neighbourhood best, for pgd, the global best, in equation (1). Early experience (again mainly trial and error) led to neighbourhood sizes of about 15 percent of the population size being used for many applications. So, for a population of 40 particles, a neighbourhood of six, or three topological neighbours on each side, was not unusual. The population size selected was problem-dependent. Population-sizes of 20-50 were probably most common. It was learned early on that smaller populations than were common for other evolutionary algorithms (such as genetic algorithms and evolutionary programming) were optimal for PSO in terms of minimizing the total number of evaluations. (Population size times the number of generations) needed to obtain a sufficient solution.

IV. TURBO CODE INTERLEAVER OPTIMIZATION EXPERIMENTS

A set computational of experiments was conducted in order to evaluate genetic algorithms and particle swarm for turbo code interleaver optimization. The performance of turbo code interleaver optimized by genetic algorithms [10] and the performance of turbo code interleaver optimized by particle swarm were investigated. An experimental and simulation framework in MATLAB was used to evaluate the discussed interleaver optimization methods.

A. Optimization by genetic algorithms

The optimization based on genetic algorithms utilized the higher level chromosome genetic algorithm (HLCGA) [9]. The HLCGA is a recent GA based approach to combinatorial optimization problems. It enables the usage of any cross over operator in evolutionary search in given problem domain. In HLCGA, the population of candidate solutions is divided into several groups' higher level chromosomes (HLCs) that act on the HLCGA level as traditional chromosomes and the underlying primitive chromosomes act as genes (i.e., the cross over and mutation is applied on HLCs and problem specific functions are used to transfer the effect of the operator on the primitive chromosomes assigned to the particular HLC). The fitness function used in this experiment was based either on the measurement of BER after simulated data transmission or on an analytical estimation of BER based on free distance. Its goal was to maximize dfree and minimize Nfree and wfree. The fitness function f was defined as follows:

$$f = A.\ dfree - B.\ Nfree - C.\ wfree \quad (4)$$

where the coefficients A, B and C were initially fixed to 100, 10 and 1 respectively. We have genetically evolved interleavers for 64-bit to 1024-bit interleavers the settings for all optimization experiments were as follows:

- HLCGA with elitary selection and study state migration
- Below 100 generations
- Probability of crossover 0.8
- Probability of mutation 0.2
- Population size is 40
- Fitness criteria was minimal BER after simulated submission of 100 random frames of weight up to 6 on the AWGN channel and dfree based fitness function as defined in (4)
- The final evaluation of obtained interleaver was done over AWGN channel for $\frac{Eb}{N0} \in [0,4]$

B. Optimization by particle swarm

The experiments using particle swarm evolution focused on its comparison with GA, in the turbo code interleaver optimization area. The settings for PSO in this experiment were [14]:

- Random selection of particles
- Below 100 iterations
- Population size is 40
- c1 and c2 equal to 2.0
- rand() and Rand() equal to 1.0
- Fitness value can be find from the velocity and position equations (2), (3)

The results of experimental interleaver optimizations are summarized in Table I, Table II and Table III.

TABLE I: COMPARISON OF AVERAGE FREE DISTANCES OF A RANDOM INTERLEAVER, INTERLEAVER EVOLVED BY GA AND INTERLEAVER EVOLVED BY PSO

N	avg. random	avg. GA	avg. PSO
64	13.50	15.70	17.20
128	14.25	17.92	18.50
256	15.75	20.20	19.10

TABLE II: COMPARISON OF BEST FREE DISTANCES OF A RANDOM INTERLEAVER, INTERLEAVER EVOLVED BY GA AND INTERLEAVER EVOLVED BY PSO

N	max. random	max. GA	max. PSO
64	13	17	17
128	14	19	19
256	17	21	20

TABLE III: COMPARISON OF BER'S FOR GA AND PSO WITH RESPECT TO $\frac{Eb}{N0}$ FOR 10 ITERATIONS

$\frac{Eb}{N0}(dB)$	BER (GA)	BER (PSO)
1	0.220	0.180
2	0.070	0.050
3	0.040	0.030
4	0.045	0.035
5	0.035	0.030
6	0.035	0.030
7	0.035	0.030
8	0.030	0.025
9	0.030	0.025
10	0.030	0.025

The results of experimental interleaver optimizations are summarized in Table I, Table II and Table III. Table I outlines the average obtained interleaver free distance for 64-, 128-, 256-bit interleavers. The second column shows average dfree of an interleaver optimized by random and third, fourth columns show average dfree of an interleaver optimized by GA and PSO respectively [13]. The experiments confirmed that all the two algorithms are

powerful interleaver optimizers. PSO evolution provided better average free distance for 64, 128 and 256-bit interleavers. Table II outlines the maximum obtained interleaver free distance for 64-, 128-, 256-bit interleavers. The second column shows maximum dfree of an interleaver optimized by random and third, fourth columns show maximum dfree of an interleaver optimized by GA and PSO respectively. The experiments confirmed that all the two algorithms are powerful interleaver optimizers. PSO evolution provided better maximum free distance for 64, 128 and 256-bit interleavers. Table III outlines the efficient BER obtained for GA and PSO. The experiments confirmed that the efficient BER can be obtained by PSO rather than GA.

V. RESULTS

The experimental optimization results are summarized in Fig.3, Fig.4, Fig.5, Fig.6 and Fig.7

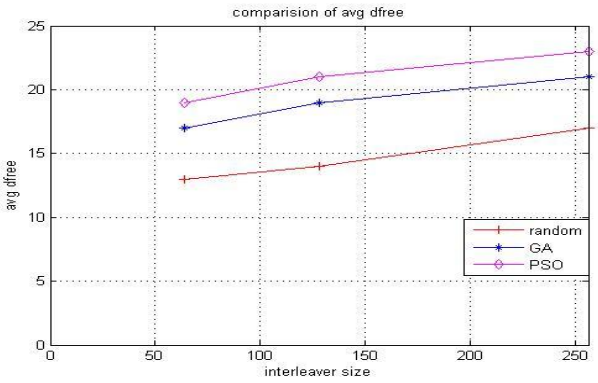

Fig.3 interleaver size versus avg. dfree

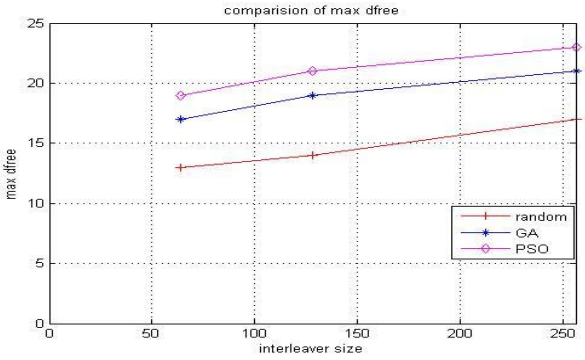

Fig.4 interleaver size versus max. dfree

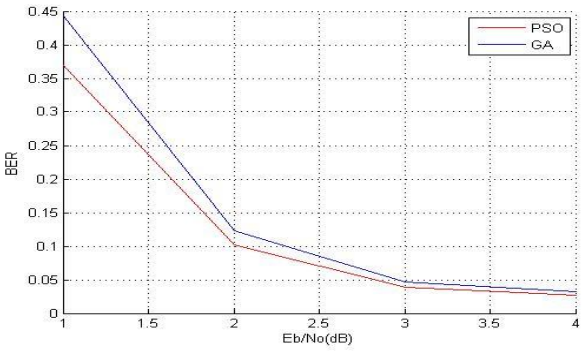

Fig.5: BER versus $\frac{Eb}{N0}$ for 4 iterations

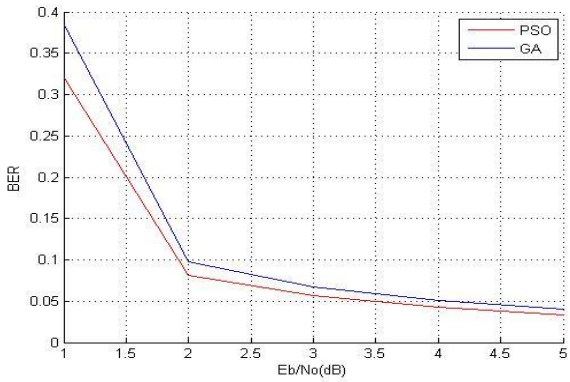

Fig.6: BER versus $\frac{Eb}{N0}$ for 5 iterations

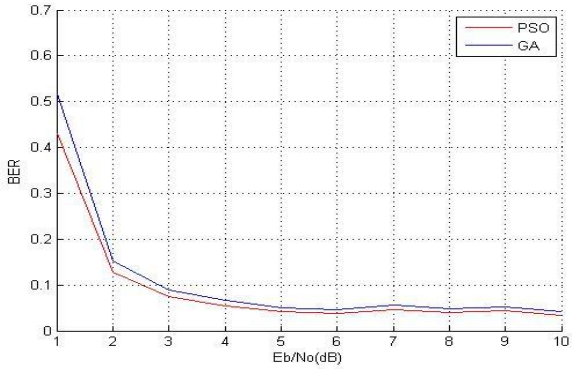

Fig.7: BER versus $\frac{Eb}{N0}$ for 10 iterations

VI. CONCLUSIONS

The effective free distance to lower the error floor seems to be good criteria to improve. It was observed that higher the free distance, better the performance. This project discussed the solution to improve the free distance as the turbo code interleaver has to be optimized by using PSO and Genetic algorithms. Moreover, these algorithms are not limited by the size and higher interleaver sizes are already being tested. An analytical estimation of free distance was used as a basis of fitness function for GA and PSO. By comparing the performance in terms of free distance and Eb/N0 Vs BER, they illustrated that PSO performs well and also the number of iterations required for the convergence in PSO is less compared to GA. 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.

REFERENCES

[1] 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.

[2] B.Vucetic and J.Yuan, "Turbo codes, Principles and applications", Kluwer Academic, Boston; London, 2000.

[3] J. G. Proakis, Digital Communications. New York: Mc Graw-Hill, 4th ed., 2001.

[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. Mitchell, An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1996
[6] A. Czam, C. MacNish, K. Vijayan, and B. A. Turlach, "Statistical exploratory analysis of genetic algorithms: The influence of gray codes upon the difficulty of a problem," in Australian Conference on Artificial Intelligence (G. I. Webb 677 and X. Yu, eds.), vol. 3339 of Lecture Notes in Computer Science, pp. 1246-1252, Springer, 2004.
[7] R. K. Ursem, "Models for evolutionary algorithms and their applications in system identification and control optimization." University of Aarhus, Denmark, April 2003.
[8] 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. Back, ed.), (San Francisco, CA), pp. 362-369, Morgan Kaufmann, 1997.
[9] P.Kromer, V.SmiSel, J.Platos, and D.Husek, "Genetic Algorithms for the Linear Ordering Problem," Neural Network World, vol. 19, no. 1, pp. 65-80, 2009. ISSN 1210-0552, impact factor 0.395.
[10] N. Durand, J. Alliot, and B. Bartolome, "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.
[11] J. Kennedy and R. Eberhart (1995) "Particle Swarm Optimization" Proceedings of the 1995 IEEE International Conference on Neural Networks (Perth, Australia): IEEE Service Center, Piscataway, NJ, IV: pp 1942-1948.
[12] R. Eberhart and J. Kennedy (1995) "A New Optimizer using Particle Swarm Theory", Proceedings of the Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan): IEEE Service Center, Piscataway, NJ: pp 39-43.
[13] J. Vesterstrøm and R. Thomsen (2004) "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems", Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Volume 2, pp. 1980 - 1987
[14] R. Eberhart and Y.Shi, 1998, "Comparison between Genetic Algorithms and Particle Swarm Optimization", EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII, pp. 611- 616.

