Proceedings of the 6th Nordic Signal Processing Symposium - NORSIG 2004
June 9 - 11, 2004
Espoo, Finland

Designing Multiplicative General Parameter Filters Using Multipopulation Genetic Algorithm

Jarno Martikainen[1] and Seppo J. Ovaska[2]

Helsinki University of Technology
Institute of Intelligent Power Electronics
P.O. Box 3000 FIN-02015 HUT
FINLAND
[1]E-mail: jkmartik@cc.hut.fi
[2]E-mail: ovaska@ieee.org
URL: http://www.hut.fi/Units/PowerElectronics/

ABSTRACT

Multiplicative general parameter (MGP) approach to finite impulse response (FIR) filtering offers a convenient way to realize cost effective adaptive filters in compact very large scale integrated circuit (VLSI) implementations used for example in mobile devices. MGP-FIRs have been designed before using single population genetic algorithms (SPGA), but in this paper we introduce a multipopulation genetic algorithm (MPGA) and apply it to the design process of MGP filters. MPGAs are usually able to search the solutions space more efficiently without getting stuck at local maxima while simultaneously converging more rapidly than traditional SPGAs. MPGAs are typically run on a parallel architecture, but the MPGA presented in this paper is intended to be used like a single population GA running on a single processor architecture.

1. INTRODUCTION

Predictive lowpass and bandpass filters play an important role in numerous delay-constrained signal processing applications, especially in the area of 50/60 Hz power systems instrumentation. In these applications distorted line voltages or currents should be filtered without delaying the fundamental frequency component. Since the line frequency tends to vary within a constrained interval, typically ±2%, the used filter should have adaptive capabilities. Vainio and Ovaska introduced the multiplicative general parameter (MGP) finite impulse response (FIR) filtering scheme in [1] and [2].

In MGP-FIR the adaptation is achieved through adjusting the two MGPs. The coefficient values of the basis filter do not change during the adaptation process. An MGP-FIR is designed in the following way. First we optimize the fixed basis filter either by applying traditional hard computing methods, or genetic algorithms (GA) [3][4]. Next, the MGP-FIR is used in the actual application, and fine-tuning is left to the multiplicative general parameters. The better the basis filter attenuates unwanted disturbances and the better the prediction capabilities of the basis filter are, the easier it is for the MGP algorithm to adapt to the changing characteristics of the input signal.

Multipopulation GAs (MPGA) [6][7] differ from single population GAs (SPGA) in that MPGAs consist of multiple populations that evolve in parallel as opposed to SPGAs where only a single population is used at a time. Such an MPGA based approach is appealing for use with parallel computer architectures. However, population parallelism can also be used with single processor architecture and thus it is possible to take advantage of the fast convergence and thorough search properties of MPGAs in a single processor environments. In this paper, we introduce a single processor based MPGA and use it for optimizing the MGP basis filter.

The paper is organized as follows: Section 2 describes the multiplicative general parameter approach to adaptive filtering. Section 3 explains the basics of genetic algorithms and introduces the reference GA used in this paper. Section 4 introduces the single processor based MPGA. Section 5 presents the results of optimizing the MGP basis filter using single processor MPGA and Section 6 gives concluding remarks.

2. MULTIPLICATIVE GENERAL PARAMETER

The MGP filtering method sets low computational requirements for the implementation platform, while it simultaneously provides effective, delayless, and robust filtering of disturbances. In a typical MGP-FIR, the filter output is computed as

$$y(n) = g_1(n)\sum_{k=0}^{N-1} h_1(k)x(n-k) + g_2(n)\sum_{k=0}^{N-1} h_2(k)x(n-k). \quad (1)$$

Where $g_1(n)$ and $g_2(n)$ present the adaptive MGP parameters, and $h_1(k)$ and $h_2(k)$ are the fixed coefficients of an FIR basis filter. Thus, the coefficients of the composite filter are $\theta_1(k) = g_1(n)h_1(k),\ k \in [0,1,...,N-1]$, for the first MGP, and $\theta_2(k) = g_2(n)h_2(k),\ k \in [0,1...,N-1]$, for the second MGP. An example of MGP-FIR with N=4 is shown in Fig. 1. Here N denotes the filter length. The adaptive coefficients are updated as follows

$$g_1(n+1) = g_1(n) + \mu e(n)\sum_{k=0}^{N-1} h_1(k)x(n-k) \quad (2)$$

$$g_2(n+1) = g_2(n) + \mu e(n)\sum_{k=0}^{N-1} h_2(k)x(n-k), \quad (3)$$

where μ is the adaptation gain factor and $e(n)$ is the prediction error between the filter output and the training signal, i.e. $(x(n) - y(n-p))$, p being the prediction step. The MGP-FIR has two adaptive parameters to adapt only to the phase and amplitude of the nominal frequency. More degrees of freedom would allow the filter to adapt also to undesired properties, such as the harmonic frequencies.

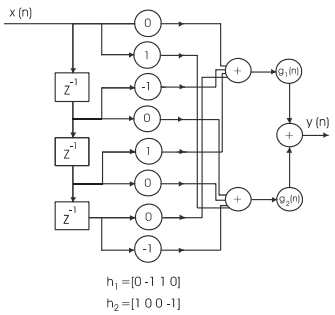

Fig. 1. An example of MGP implementation, where N=4. Signal values $x(n-1)$ and $x(n-2)$ are connected to the first MGP and values $x(n)$ and $x(n-3)$ are connected to the second MGP with filter coefficients -1, 1, 1, and -1 respectively

A training signal, whose length is 900 samples, is used in this paper. This signal is divided into three parts, each of which contains 300 samples. These training signal blocks correspond to frequencies of 49 Hz, 50 Hz, and 51 Hz. The training signal constitutes thus of the nominal frequency sinusoid with amplitude of 1, odd harmonics up to the 15th with amplitudes 0.1 each, and white noise. The training signal is similar to that used in [1] and [2], and it mimics the line signal (voltage/current) with varying fundamental frequency, harmonics, and noise.

The basic idea of MGP-FIR filters is that all the samples of input delay line should be connected either to the first or to the second MGP, and no value should be left unused. Computational efficiency of these particular MGP filters arises from the fact that the filter coefficients are either -1, 0, or 1. Thus the number of multiplications in filtering operations is radically reduced compared to a normal filtering operation using more general coefficient values.

3. GENETIC ALGORITHMS

Genetic algorithms model the nature's way of optimizing the characteristics of a system to comply with the demands of the surrounding environment.

In this application, the cross connections and coefficients of the basis filter are to be optimized using a GA. These filter tap cross connections and coefficients are modeled as chromosomes in the following way. There are two separate vectors in one chromosome: the first vector, $h_1(n)$, of the chromosome represents the weights of the input samples connected to the first MGP, and the second vector, $h_2(n)$, corresponds to the weights of the input samples connected to the second MGP. The weights can be -1, 0, or 1. -1 and 1 in a vector indicate that this particular input value is associated to the corresponding MGP. 0, on the other hand, determines that a particular input value is not associated with this MGP, rather it is connected to the other MGP.

In [2] Ovaska and Vainio stated that satisfactory results can be achieved using an initial population, the size of which is twice the number of the filter length. Since the filter length N in our case is 40, the initial population size was set to be 80.

The fitness of the chromosomes is evaluated using the fitness function:

$$f = \frac{K}{\max(NG49, NG50, NG51)\cdot(ITAE49 + ITAE50 + ITAE51)}. \quad (4)$$

K is assigned a value of 10^7 to scale the output of the fitness function to be expressed in thousands. Terms $NG49$, $NG50$, and $NG51$ represent the white noise gain at a specific stage of the test input signal. These terms were added to the fitness function to provide noise attenuation. The white noise gain is calculated as

$$NG(n) = \sum_{k=0}^{N-1}\left[g_1(n)\cdot h_1(k)\right]^2 + \sum_{k=0}^{N-1}\left[g_2(n)\cdot h_2(k)\right]^2. \quad (5)$$

$g_1(n)$ and $g_2(n)$ represent the first and second MGP at the end of a certain frequency period, respectively, whereas $h_1(n)$ and $h_2(n)$ denote the filter coefficients associated to the corresponding MGPs. In other words, $NG49$ is calculated using the converged MGP values after 300 samples, $NG50$ and $NG51$ are calculated after 600 and 900 samples, respectively,

the frequency of the training signal changing every 300 samples.

*ITAE*49, *ITAE*50, and *ITAE*51stand for the Integral of Time-weighted Absolute Error (ITAE) for each of the three signal parts, respectively. These terms were added to the fitness function to smoothen the adaptation of the filter to the varying input signal characteristics. The ITAE is calculated as

$$ITAE = \sum_{n=1}^{M} n \cdot e_f(n). \quad (6)$$

n is the sample index, and M stands for the sample number at the end of a specific frequency period, in this case 300, 600 and 900. $e_f(n)$ is the error between the output of the system and the pure input signal without any harmonics or noise for a single frequency period.

In this paper, both the SPGA and the MPGA given 1000 generations to converge to a solution, and both the genetic algorithms were run 20 times to get some statistical reliability. A single simulation run of 1000 generations using initial population of 80 chromosomes and filter length of 40, took 70-80 minutes to run on a Dell computer equipped with 2.66 GHz Intel Pentium 4 processor, 512 MB of memory, and MATLAB 6.5.0 software.

3.1 Single Population Genetic Algorithm

The SPGA used as reference in this paper was structured as follows. The population size was constant at 80. Mating was applied based on the rank of the chromosomes so that the best chromosome would mate with the second best, the third with fourth and so on. 40 best chromosomes would produce 40 offspring during each generation. Crossover and mutation used adaptive probabilities as described in [5]. The observed structure of the converged solution suggested that the genes tend to cluster in the chromosomes, so that a varying number of the same gene value would appear consecutively. This was taken into account while generating the initial population, instructing the GA to create initial chromosomes, in which the genes were clustered.

4. MULTIPOPULATION GENETIC ALGORITHM

MPGAs are usually associated with parallel computer architectures [6][7]. In these traditional MPGAs several SPGAs are run in parallel using a variety of migration procedures to swap chromosomes between different processors and thus populations. However, the chromosome population can be divided advantageously into separate entities even though only a single processor is available for the genetic algorithm.

Our approach to MPGA using a single processor originates from the notion that whether it being a human or animal population consisting of only a few or millions of individuals, the population tends to divide at least into two sub populations based on evaluation of some, not always so clear, fitness function. In this paper, we divide the initial population into two sub populations calling the populations the “elite population” and the “plain population”. In the following the procedure of our MPGA is explained step by step.

Step 1: The MPGA starts using a single population of 80 chromosomes. Seeding similar to that used in the reference SPGA was used to create the initial population. The sum on the elite and plain population sizes remains constant throughout the optimization process, but the sub population sizes vary in time. In the beginning, the elite population size is 0 and the plain population size is 80. Until the conditions for creating the elite population are met, the MPGA works like a SPGA.

Step2: We start to build the elite population after one of the three conditions is fulfilled: there exists a chromosome with fitness value of 2400 or more, the GA has the same maximum fitness value for more than five consecutive generations, or the simulation has run for 20 generations. The elite population is composed as follows: two best chromosomes with different fitness values from the plain population are added to the elite population if the fitness value of the best plain population chromosome is more than 95% of that of the best chromosome in the elite population.

Step 3: As soon as the creation of the elite population is started, the elite chromosomes start to mate with each other. In plain population the mating is based purely on the fitness values, so that the best chromosome mates with the second best, the third one mates with fourth, and so on. In the elite population the parents are randomly chosen from the best quarter so that a chromosome cannot mate with itself.

The next generation elite population consists of the parents of the previous generation, the offspring and possibly two new chromosomes from the plain population. The elite chromosomes that do not fit to the elite population in the next generation are transferred back to the plain population. The plain population consists of parents, the offspring and possibly chromosomes from the elite population. In both the elite and the plain population half the chromosomes in the population act as parents for the offspring. When a steady state is reached, that is the elite population size is 10% of the total chromosome number four chromosomes in the elite population produce four offspring and 36 chromosomes in the plain population produce 36 offspring.

Step 4: Elitism is applied in the mutation scheme. Before the stable state is achieved, the chromosomes in the plain population are mutated with the probability of 40%. After the elite

population has reached its final size, the chromosomes in the plain population are mutated with the probability of 80% and the chromosomes in the elite population are mutated with the probability of 40%. Mutation may change only a single gene of a chromosome at a time. From here we go to Step 2 until a satisfactory solution is found.

5. RESULTS

Table 1 presents the comparison of the performance of the MPGA and the SPGA. MPGA succeeded in achieving both faster convergence and better average fitness. The lowest fitness achieved using SPGA was 2982 the best value being 4025 and standard deviation being 329. For MPGA the corresponding values are 3673, 4032, and 104, respectively. So, it is likely that even a single run using MPGA would produce a good result. SPGA could require multiple runs until a satisfactory solution is found.

Table 1. Comparison of SPGA and MPGA

Algorithm	SPGA	MPGA
Average fitness	3618	329
Standard deviation	329	104
Convergence (generations)	155	131
Time (minutes)	11.6	9.8

Fig. 3 shows the difference between chromosomes with different fitness values. The figure is created after the 49 Hz and the 50 Hz sections of the training signal have passed. The overall fitness is calculated over all the three sections. Fig. 3 shows how all the filters try to attenuate the harmonic frequencies (150 Hz, 250 Hz, …, 950 Hz).

Fig. 4 displays the error signal between the optimal filtering result and filtering results using different chromosomes. Green signal in Fig. 4 is for comparison purposes and it presents a typical converged chromosome achieved using a very simple genetic algorithm without any adaptive probabilities or other enhanced functionality.

6. DISCUSSION AND CONCLUDING REMARKS

The presented MPGA outperformed the reference SPGA in both average convergence speed and average best fitness value. The proposed method is the first time this kind of artificial life approach is implemented in the optimization process of MGP-FIRs. Our MPGA has the capabilities of being a competitive tool for practicing engineers, since it reliably produces high fitness value chromosomes and thus requires only a small amount of runs. The proposed MPGA algorithm is useful especially in optimization problems where the fitness calculation is very time consuming, as in this case.

Our future work will concentrate on finding out the precise effects of the various parameters present in the MPGA model, such as the elite population size, migration interval, and migration conditions.

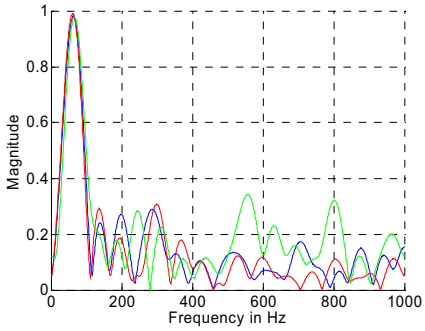

Fig. 3. Converged frequency response of the MGP fir for the 50 Hz section of the test signal. Blue uses the best chromosome achieved using MPGA (fitness 4032), red uses the worst chromosome achieved using MPGA (fitness 3673). Green is for comparison purposes (fitness 2480)

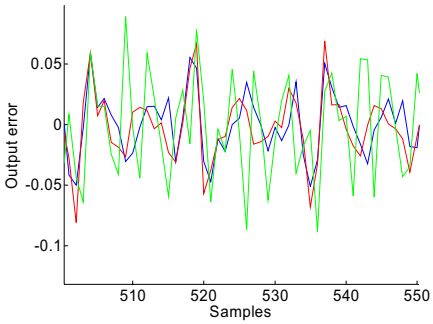

Fig. 4. Blue is the error between the ideal filtering result and filtering result using the best chromosome achieved using MPGA (fitness 4032). Red is the error between the ideal filtering result and the filtering result using worst chromosome achieved using MPGA (fitness 3673). Green uses chromosome with fitness value of 2480

7. REFERENCES

1. Vainio, O., Ovaska, S.J., Pöllä, M.: Adaptive filtering using multiplicative general parameters for zero-crossing detection. IEEE Transactions on Industrial Electronics, Vol. 50, No. 6, Dec 2003, 1340-1342
2. Ovaska, S.J, Vainio, O.: Evolutionary programming in the design of adaptive filters for power systems harmonies reduction. IEEE International Conference on Systems, Man, and Cybernetics, Vol. 5, Oct 2003, 4760-4766
3. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, NY (1996)
4. Haupt, R. L., Haupt, S. E.: Practical Genetic Algorithms. John Wiley & Sons, New York, NY (1998)
5. Srinivas, M., Patnaik, L. M.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 24, No. 4, Apr 1994, 656-667
6. Kim, J., Khosla, P.K.: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992, Vol. 1, Jul 1992, 279 - 286
7. da Silva, J.D., Simoni, P.O.: The island model parallel GA and uncertainty reasoning in the correspondence problem. Proceedings of the International Joint Conference on Neural Networks, 2001, Vol. 3, Jul 2001, 2247 - 2252