

Jitter Minimization with Genetic Algorithms

Fernanda Coutinho
Instituto Superior de Engenharia de Coimbra
Quinta da Nora
3030 Coimbra, Portugal
fernanda.coutinho@isec.pt

José A. Fonseca
Universidade de Aveiro / IEETA
Depto de Electrónica e Telecomunicações
3810 Aveiro, Portugal
jaf@det.ua.pt

Jorge Barreiros
Centro de Informática e Sistemas
Universidade de Coimbra, Polo II
3030 Coimbra, Portugal
jmsousa@isec.pt

Ernesto Costa
Centro de Informática e Sistemas
Universidade de Coimbra, Polo II
3030 Coimbra, Portugal
ernesto@dei.uc.pt

Abstract

Transmission network induced jitter in periodic control variables is a known problem on fieldbus based distributed systems for embedded control applications. This jitter can be reduced or eliminated if adequate release instants are imposed to the periodic messages transmitted. In this paper, jitter reduction is achieved using a variant genetic algorithm that determines an adequate initial phasing. This algorithm can be used progressively, i.e., it can work first on a subset of the messages, thus finding solutions for the higher priority ones, and later including the other messages until the whole set is considered. Experimental results obtained with two well-known and widely used benchmarks, the PSA, coming from automotive industries, and the SAE from Automatically Guided Vehicles, show complete elimination or a significant decrease of jitter when compared to non-optimized systems.

1. Introduction

Distributed systems are today widely used in control applications. In these, control tasks such as data acquisition, control algorithm execution, system identification, actuation, are often carried out in different nodes of the system. While in stand-alone classical controllers the sampling period can usually be kept as close as possible to the desired value, in distributed systems this may not be true due to the need to transmit data over a shared communication medium.

Most distributed systems either industrial or embedded, rely on a serial communications infrastructure known as fieldbus [1] to interconnect a set of nodes. Some sort of arbitration is then used to decide, in each time instant, which information will be

transmitted on the bus [2]. When periodic variables such as control parameters are to be transmitted, it is usually possible to impose an adequate average transmission period. However, due to the interaction of the different variable periods and, often, of sporadic or aperiodic traffic, it is rather difficult to obtain constant time intervals between successive instances of the same periodic variable. The variations of the period are called jitter and instantaneous values can be defined for each instance k of each parameter i transmitted in a correspondent message:

$$j_{i,k} = t_{s_{i,k}} - (k * T_i + \Phi_i)$$

In the previous expression $t_{s_{i,k}}$ is the start time of the transmission of the instance k of message / parameter i , T_i is the period and Φ_i is the initial phasing at the start-up of the system.

The overall system jitter (OSJ) defined as:

$$OSJ = \sum_{i=1}^M \sum_{k=1}^{\frac{I}{T_i}} j_{i,k}$$

can be used as an integral measure of the problem. In this case this measure is considered in a time interval of interest, I , equal to the Least Common Multiple (LCM) of the periods T_i of the M periodic messages included in the set.

The problem of jitter in periodic control variables has been already identified in [3] where several scenarios of this problem were studied. That was the case when more than one sample occurs in the same period which leads to data rejection and when there is no sample in that interval which is known as vacant sampling. More recently, in [4] the degradation of performance in feedback control loops subject to jitter in the sampled and in the actuation variables (called read-in and read-out jitter) was analyzed. In [5] the effect of jitter due to message transmission in CAN – Controller Area Network [6] is shown to affect the phase margin of a

control loop. These and other examples demonstrate the need for further research in jitter minimization as it is pointed in [7].

In this work, it is proposed the use of genetic algorithms to reduce network induced jitter in control variables transmitted on a distributed embedded system. A variant algorithm, which we called Progressive Genetic Algorithm (proGA), is also proposed as a means to obtain progressively better solutions for the problem. Its performance is compared with the simple genetic algorithm. Moreover the variant algorithm is also well suited for application in a real-time on-line jitter optimization device as intermediate solutions can be used during the process.

This paper is structured as follows: next section contains a brief overview of how this technique can be used in fieldbuses; in section 3 the genetic algorithms used are introduced and the modeling of the problem is presented; in section 4 the experiments conducted on two message sets usually used as benchmarks for the automotive industry are described and the results are briefly analyzed; finally, in section 5, some conclusions are presented and a short discussion on future work is made.

2. Applying jitter minimization in fieldbuses

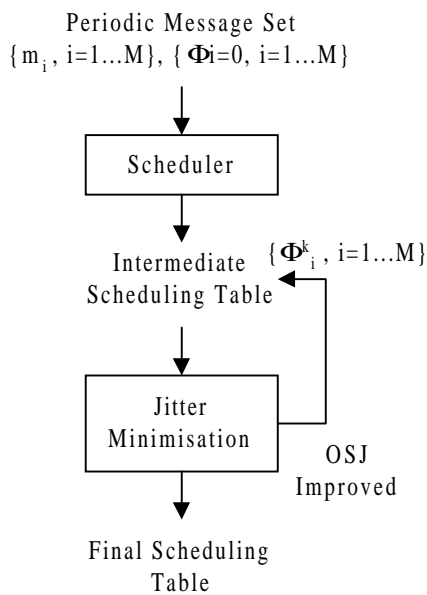


Figure 1. Applying the jitter minimization algorithm

The jitter minimization technique here proposed can be directly applied in every off-line scheduled fieldbus following the procedure depicted in figure 1. In fact, its application just requires that the initial scheduling table can be worked by the algorithm in order to change the initial phasing Φ_i of the periodic messages. The

scheduling algorithm used can build the initial table considering the simultaneous release of the first instance of all the periodic messages in the set ($\Phi_i = 0$). The jitter minimization algorithm will generate successive intermediate scheduling tables in which changes in the messages initial phasing were imposed. One of these tables will become definitive when no further improvement is found in the OSJ value.

Examples of common fieldbuses where this technique can be applied are FIP [8], TTP [9] any time triggered systems in general. The future standardization of time-triggered CAN (ISO TC22/ SC3/ WG1/ TF6) will then be a candidate to apply the technique during the scheduling of the time slots in which each node will transmit its periodic data.

As it is explained later, a variant of the algorithm called proGA can be applied starting the jitter minimization in a subset of the messages, in principle the ones with higher transmission priority. After, all the other messages will be considered if required. This feature opens the possibility to apply the algorithm on-line, supplying the successive partial results to a communications dispatcher. In this case, the communication system must support on-line changes to the set of periodic messages. FTT-CAN [10] is an example of such a system where the use of this technique is presently being considered.

3. Progressive Genetic Algorithm

3.1. Genetic Algorithms

Genetic Algorithms (GA) are stochastic search procedures inspired by the biological principles of natural selection and genetics [11] [12]. In spite of their possible variants, GA can be described by the following general procedure:

Procedure GA

```

t=0;
Initialize P(t);
Evaluate P(t);
While stoping_criterion_false do
  t = t+1;
  P'(t) = select_from P(t-1);
  P''(t) = use_op_modification P'(t);
  Evaluate P''(t);
  P(t) = merge P''(t), P(t-1)

```

End_do

The GA starts with a set of candidate solutions called a population, usually defined randomly. Each element of that initial population, called an individual, is then evaluated using a fitness function that gives a measure of the quality of that element. Each individual is in fact an aggregate of smaller elements or units, which are called genes. Each gene can have different values or alleles. The algorithm enters then a cycle in order to generate a new population. It starts by probabilistically selecting the

fittest individuals. Then they undergo a modification process, using genetic inspired operators like crossover or mutation that will eventually alter the alleles of some genes. Finally, the old and new populations are combined and the result becomes the next generation that will in turn be evaluated. The cycle stops when a certain condition is achieved (for instance, a pre-defined number of generations). The algorithm just described generally works with a low-level representation of each individual called its genotype. Nevertheless, in complex problems, the fitness function acts upon a high-level representation of an individual, its phenotype, making necessary to use decoders from genotypes to phenotypes.

The success of GA algorithms is linked to their ability to solve difficult problems: problems where the search space is large and multi-modal. This is the case of the problem described above where the goal is to minimize the overall message transmission jitter in a priority-based real-time communication system. But for a GA to be more efficient than traditional algorithms it is crucial to incorporate in it problem-specific knowledge. This is achieved by choosing a good data structure to represent the chromosome and also by “tuning” the genetic operators [13].

3.2. The elements of a simple GA (sGA)

For this particular problem, the genotype will be a vector of integers where each integer represents the initial offset of one message. For a message M each integer will be within the range of $[0, T_M - 1]$, where T_M is the period of that message. All the phases for which the previous condition holds true are said to be *feasible*.

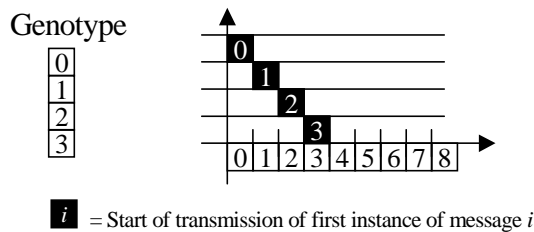


Figure 2. Phases representation.

In GAs the crossover operator mimics the one that occurs in natural sexual reproduction. Crossover uses two individuals of the population (the *progenitors*) to generate two new individuals (the *descendants*). In this case it is used a *one-point* crossover. Thus, a single random crossover point R is chosen, within the range of 0 and M (where M is equal to the number of messages). For the phases between 0 and R, the resulting descendent is equal to one of the progenitors. For phases between R+1 and M, it is equal to the other progenitor. A second descendent is created inverting the order of the progenitors. In this system, crossover probability is 1. This means that all the descendants are created with the crossover operator.

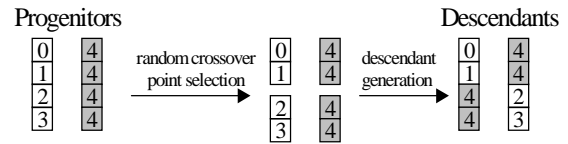


Figure 3. One-point crossover.

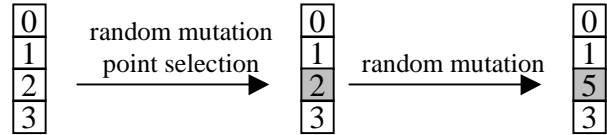


Figure 4. Mutation operation.

In this application of GAs it is also used a simple mutation (see Figure 3 above) in which one individual’s gene is changed probabilistically. The probability used is 3%.

The fitness of each individual is evaluated by running a simulation of the system operation that calculates the OSJ which is the sum of the partial jitter of each message as defined in section 1. The fittest individuals are those for which this value is smaller. The simulation process will be described later.

The genetic algorithm stops when a predetermined number of consecutive descendants are not included in the population. This happens when they are all worse than every individual already on the population.

The progenitors are selected proportionally to their fitness.

At each iterative step, one or two of the worst individuals of the old population may be replaced by one or both of the generated descendants. In this way, a new population is generated. This replacement takes place when the fitness of the later is better than the fitness of the former (steady state like GA mechanism). This will in principle improve the average fitness of the population.

The initial population is randomly generated but constrained to feasible individuals.

3.3. The Simulator

Fitness evaluation is done by simulating system execution for an overall system period. This period is equal to the LCM (least common multiple) of the periods of the messages. It was found that the best performance, amongst several which were developed and tested, was attained with an event-driven simulator.

This simulator uses two integer vectors for holding information about the system’s state: the elapsed transmission time of each message and the next release time of each message. This information is combined to determine the instants where the system’s state should be examined. By analysing and updating the system’s state at these points, it is possible to determine the overall jitter within a reasonable time for practical systems.

Condition	OSJ
Simultaneous release of messages	3679408
Best of 5000 random phase sets (1 μ s res.)	103613
Best of 5000 random phase sets (10 μ s res.)	103524
Best of 5000 random phase sets (100 μ s res.)	110900

Table 1. Jitter values for not optimized systems (PSA at 125 Kbit/s).

Condition / Resol.	1 μ s	10 μ s	100 μ s
Lowest OSJ	12640	12640	22640
Average OSJ	13947	133645	14619
Worst OSJ	16256	15272	16304
Exec. Time Average	160	171	124
N° of Experiences	20	20	20

Table 2. Summary of optimization results for PSA benchmark at 125 Kbit/s.

Variable / Resol.	1 μ s	10 μ s	100 μ s
1-engine controller	0	0	0
2-wheel angle sensor	1200	3170	13200
4-AGB	6778	12740	1800
7-device x	1800	2720	11800
3-engine controller	5956	16320	15900
5-device x	15909	6060	4000
9-device y	8221	11970	14300
6-device x	18383	21900	5900
8-bodywork sensor	14072	14260	28200
11-AGB	3976	24280	8300
10-engine controller	24047	4190	18200
12-device x	12686	28640	22600

Table 3. Best phasing set for PSA benchmark variables at 125 Kbit/s.

Condition	OSJ
Simultaneous release of messages	1764704
Best of 5000 random phase sets (1 μ s res.)	5962
Best of 5000 random phase sets (10 μ s res.)	5544
Best of 5000 random phase sets (100 μ s res.)	5488

Table 4. Jitter values for not optimized systems (PSA at 250kpbs).

Condition / Resol.	1 μ s	10 μ s	100 μ s
Lowest OSJ	0	0	0
Average OSJ	0	0	0
Worst OSJ	0	0	0
Exec. Time Average	28	28	28
N° of Experiences	20	20	20

Table 5. Summary of optimization results for PSA benchmark at 250 Kbit/s.

Condition	OSJ
Simultaneous release of messages	8435636

Best of 5000 random phase sets (1 μ s res.)	156783
Best of 5000 random phase sets (10 μ s res.)	163546
Best of 5000 random phase sets (100 μ s res.)	143952

Table 6. Jitter values for not optimized systems (SAE at 250 Kbit/s).

Condition / Resol.	1 μ s	10 μ s	100 μ s
Lowest OSJ	0	0	0
Average OSJ	68	178	545
Worst OSJ	230	1374	1964
Exec. Time Average	540	739	928
N° of Experiences	20	20	20

Table 7. Summary of optimization results for SAE benchmark at 250 Kbit/s.

Variable / Resol.	1 μ s	10 μ s	100 μ s
7-Accelerator Position	0	0	0
9-Brake Pressure, Line	3975	4740	3500
8-Brake Press., Master Cyl	1225	3910	4400
32-Clutch Press. Control	4382	4190	4100
49-Proc. Motor Speed	4745	2080	4700
42-Torque Command	633	3390	3800
43-Torque Measured	1477	4460	900
11-Tran. Clutch Line Press	954	1820	1200
29-High Contactor Control	6855	5420	7400
30-Low Contactor Control	8400	3120	300
14-Hi&Lo Cont. Op/Close	10358	8000	7100
25-12V Pwr Ack I/M Cont	11772	17730	600
...

Table 8. Best phasing set for a sample of SAE benchmark variables at 250 Kbit/s.

4.2. Results Analysis

The solution given by proGA offers always a considerable reduction of jitter when compared with systems that were not optimized. Good results were obtained on all tests, showing that a complete elimination of jitter is possible on the SAE benchmark set at 250kbps, even with a coarse timer resolution of 100 μ s.

Additional experiments revealed that the proGA is more efficient than the simple GA. The proGA is more time-effective and offers better results consistently [16]. When comparing to the best random not optimized solutions, the effectiveness of the algorithm can be easily observed. Improvement ranges from around 88,8% on the worst case (PSA benchmark @ 125kbps) reduction up to the complete elimination of jitter on all other cases.

A timer resolution of 100 μ s seems to be enough to get good results out of this jitter reduction technique. In fact, the best results with that resolution were equal to results obtained with the highest resolutions in every test case.

The speed of the proGA algorithm, although comparable or superior to the simple GA's, is

insufficient for direct on-line full optimization. However, the algorithm's iterative approach is particularly well suited for implementing a best-effort on-line optimizer. The successive partial results generated by the algorithm can be supplied to a communications dispatcher thus making the system evolve to a near-optimal message phasing. This may prove to be very effective, because the optimization of the messages with higher priority occupies a very small percentage of the total execution time of the algorithm, and those are the messages with most impact on overall bandwidth occupation. The next graphic shows the temporal evolution of the algorithm for some runs on the PSA-125 benchmark. It can be easily seen that the first 6/7 messages are quickly optimized.

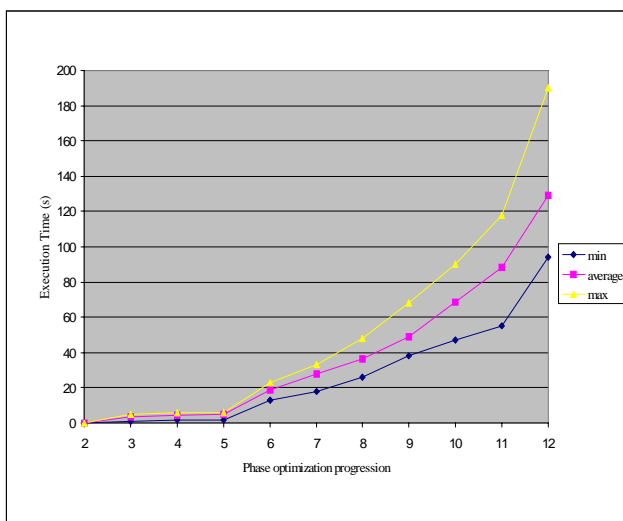


Figure 6. Evolution of the execution time of the algorithm.

4. Conclusions and Future Work

In this paper a technique for reduction of network induced jitter in message transmission, using genetic algorithms is proposed. A variant genetic algorithm (proGA - progressive GA) specifically developed for this problem is shown to provide adequate performance and on-line optimization capabilities. Additionally, the effects of variation of phasing timer resolution on minimum overall system jitter were also briefly analyzed.

The experimental results show that the algorithm has significant optimization capabilities for on-line or off-line jitter reduction. Reduction of jitter is very significant when compared with the situation without any optimization. Additional work proves that the proGA algorithm is more effective and efficient than a simple GA approach.

Although execution times are high for obtaining fully optimized solutions, near-optimal solutions are obtained

quickly, and these may be used to implement an on-line jitter reduction system, through continuous phase shifting. Additionally, it was found out that the timer resolution for the initial phases can be rather coarse. No impact on the quality of the solution was observed when lowering resolution from 1 μ s to 100 μ s.

Although the technique imposes a relatively high computational overhead, at least considering the usual processors available in these types of systems, it is possible to apply it incrementally. The initial solutions for a small (5,6) number of messages are usually obtained rather quickly, making it feasible to develop an on-line optimization device that produces best-effort solutions to a communication scheduler or dispatcher. An adequate system or coprocessor can then work on-line on messages phasing promoting a continuous reduction of jitter along system operation. This opens additional interesting possibilities for the on-line admission of new real-time messages.

References

- [1] J. P. Thomesse, "A Review of the Fieldbuses", *Annual Reviews in Control*, 22 pp. 35-45, 1998.
- [2] L. Almeida, M.L. Chavez, J. A. Fonseca, J.P. Thomesse, "Real time communications in manufacturing" *Proceedings ISAS'99 The 5th. Int'l Conference on Information Systems Analysis and Synthesis*, Orlando, USA, July/August 1999.
- [3] S. Hong, "Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems", *IEEE Transactions on Control Systems Technology*, Vol. 3, N° 2, June 1995.
- [4] Stothert, etal "Effect of Timing Jitter on Distributed Computer Control System Performance", *Proc. 15 IFAC Workshop DCCS'98 - Distributed Computer Control Systems*, September 1998.
- [5] G. Juanole, "Modélisation et Évaluation du Protocole MAC du Réseau CAN", *École d'été ETR'99 - Applications, Réseaux et Systèmes*, ENSMA, Poitiers, France, September 1999.
- [6] "CAN specification version 2.0 - Technical Report", Bosch GmbH, Stuttgart, Germany, 1991.
- [7] J.D. Decotignie, "Some Future Directions in Fieldbus Research and Development", *Proceedings FeT '99 - Fieldbus Systems and Applications Conference*, Magdeburg, Germany, September 1999.
- [8] P. Leterrier, "The FIP Protocol", WorldFip Europe, 2-4 Rue de Bône, 92160 Antony - France, 1992.
- [9] H. Kopetz, G. Grünsteidl, "TTP - A Protocol for Fault-Tolerant Real-Time Systems", *IEEE Computer*, 27(1), 1994.
- [10] L. Almeida, J.A. Fonseca "A Flexible Time-Triggered Communication System Based on the Controller Area Network" *Proceedings FeT '99 - Fieldbus Systems and*

their Applications Conference, Magdeburgo, Germany, September 1999.

- [11] J. Holland, "Adaptation in natural and artificial systems", *University of Michigan Press*, Michigan, 1975
- [12] T. Back, D. Fogel, Z. Michalewicz (eds.), "Handbook of Evolutionary Computation", Oxford University Press, New York, 1997.
- [13] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs" (3rd, revised and extended edition), Springer-Verlag, Berlin, 1999.
- [14] K. Tindell, A. Burns, "Guaranteeing Message Latencies on Control Area Network (CAN)", *Proceedings of ICC'94 (1st International CAN Conference)*, Mainz, Germany, 1994.
- [15] N. Navet, Y.-Q. Song, "Performance and Fault Tolerance of Real-Time Applications Distributed over CAN (Controller Area Network)", *CiA – CAN in Automation Research Award*, 1997.
- [16] J. Barreiros, E. Costa, J. A. Fonseca, F. Coutinho, "Jitter reduction in a real-time message transmission system using genetic algorithms", to appear in *Proceedings of the CEC 2000 - Conference of Evolucionary Computation*, USA, July 2000.