COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMISATION

Dr. Karl O. Jones

***Abstract**: In recent years the area of Evolutionary Computation has come into its own. Two of the popular developed approaches are Genetic Algorithms and Particle Swarm Optimisation, both of which are used in optimisation problems. Since the two approaches are supposed to find a solution to a given objective function but employ different strategies and computational effort, it is appropriate to compare their implementation. The problem area chosen is that of identification of model parameters (as used in control engineering).*

***Keywords**:* genetic algorithm, particle swarm optimisation, process modelling, simulation..

INTRODUCTION

In the early 1950s computer scientists studied evolutionary systems as an optimisation tool, introducing the basics of evolutionary computing. Until the 1960s, the field of evolutionary systems was working in parallel with genetic Algorithm (GA) research. When they started to interact, a new field of evolutionary programming appeared by introducing new concepts of evolution, selection and mutation. Holland [1] defined the concept of the GA as a metaphor of the Darwinian theory of evolution applied to biology. Implementation of a Genetic Algorithm (GA) begins with a population of random chromosomes. The algorithm then evaluates these structures and allocates reproductive opportunities such that chromosomes which represent a better solution to the problem are given more chance to "reproduce". In selecting the best candidates, new fitter offspring are produced and reinserted, and the less fit removed. In using operators such as crossover and mutation the chromosomes exchange their characteristics. The suitability of a solution is typically defined with respect to the current population [1]. GA techniques have a solid theoretical foundation [1], based on the Schema Theorem [2]. GAs are often viewed as function optimisers, although the range of problems to which they have been applied is broad, including: pattern discovery [3], signal processing [4], and training neural networks [5].

The implicit rules followed by the members of fish schools and bird flocks, that allow them to undertake synchronized movement, without colliding, has been studied by several scientists [6][7]. There is a general belief that social sharing of information among individuals of a population, may provide an evolutionary advantage, and there are numerous examples coming from nature to support this. This was the core idea behind the development of Particle Swarm Optimisation (PSO). The PSO method is a member of the wide category of Swarm Intelligence methods [8]. Kennedy originally proposed PSO as a simulation of social behaviour, and it was initially introduced as an optimisation method in 1995 [9]. PSO can be easily implemented and is computationally inexpensive since its memory and CPU speed requirements are low [10]. Furthermore, it does not require gradient information of the objective function being considered, only its values. PSO has proved to be an efficient method for numerous general optimisation problems, and in some cases it does not suffer from the problems encountered by other Evolutionary Computation techniques [9]. PSO has been successfully applied to a range of problems, from function optimisation to the training of neural networks. Although, while PSO typically moves quickly towards the best general area in the solution space for a problem, it often has difficulty in making the fine grain search required to find the absolute best point.

Many control system applications, such as model-based predictive control, require some form of mathematical model of the process to be controlled. In some instances, accurate models can be derived analytically through consideration of known physical processes. This approach is often appropriate for linear, deterministic, time-invariant, single-input single-output systems, where sufficient knowledge of the system processes is available. Most real-world systems, however, do not fit into this category. In particular, they are often non-linear and poorly understood. Black-box modelling, commonly known as "system identification", is often the only realistic approach available. System modelling can be decomposed into two inter-related, problems: *Selection* of a suitable model structure, and *Estimation* of the model parameters. The work presented here focuses on *Estimation*.

GENETIC ALGORITHM OPERATION

To illustrate the working process of genetic algorithm, the steps to realise a basic GA are listed:

Step 1: Represent the problem variable domain as a chromosome of fixed length; choose the size of the chromosome population N, the crossover probability P_c and the mutation probability P_m.

Step 2: Define a fitness function to measure the performance of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.

Step 3: Randomly generate an initial population of size N: $x_1, x_2, ..., x_N$

Step 4: Calculate the fitness of each individual chromosome: $f(x_1), f(x_2), ..., f(x_N)$

Step 5: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness. High fit chromosomes have a higher probability of being selected for mating than less fit chromosomes.

Step 6: Create a pair of offspring chromosomes by applying the genetic operators.

Step 7: Place the created offspring chromosomes in the new population.

Step 8: Repeat Step 5 until the size of the new population equals that of initial population, N.

Step 9: Replace the initial (parent) chromosome population with the new (offspring) population.

Step 10: Go to Step 4, and repeat the process until the termination criterion is satisfied.

A GA is an iterative process. Each iteration is called a generation. A typical number of generations for a simple GA can range from 50 to over 500. A common practice is to terminate a GA after a specified number of generations and then examine the best chromosomes in the population. If no satisfactory solution is found, then the GA is restarted [11].

PARTICLE SWARM OPTIMISATION ALGORITHM OPERATION

PSO optimises an objective function by undertaking a population-based search. The population consists of potential solutions, named particles, which are a metaphor of birds in flocks. These particles are randomly initialised and freely fly across the multi-dimensional search space. During flight, each particle updates its own velocity and position based on the best experience of its own and the entire population. The updating policy drives the particle swarm to move toward the region with the higher objective function value, and eventually all particles will gather around the point with the highest objective value. The detailed operation of particle swarm optimisation is given below:

Step 1: Initialisation. The velocity and position of all particles are randomly set to within pre-defined ranges.

Step 2: Velocity Updating. At each iteration, the velocities of all particles are updated according to:

$$\vec{v}_i = w\vec{v}_i + c_1R_1(\vec{p}_{i,best} - \vec{p}_i) + c_2R_2(\vec{g}_{i,best} - \vec{p}_i)$$

where $\vec{p}_i$ and $\vec{v}_i$ are the position and velocity of particle *i*, respectively; $\vec{p}_{i,\text{best}}$ and $\vec{g}_{i,best}$ is the position with the 'best' objective value found so far by particle *i* and the entire population respectively; *w* is a parameter controlling the flying dynamics; R_1 and R_2 are random variables in the range [0, 1]; c_1 and c_2 are factors controlling the related weighting of corresponding terms. The inclusion of random variables endows the PSO with the ability of stochastic searching. The weighting factors, c_1 and c_2, compromise the inevitable trade-off between exploration and exploitation. After updating, $\vec{v}_i$ should be checked and secured within a pre-specified range to avoid violent random walking.

Step 3: Position Updating. Assuming a unit time interval between successive iterations, the positions of all particles are updated according to:

$$\vec{p}_1 = \vec{p}_i + \vec{v}_i$$

After updating, $\vec{p}_i$ should be checked and limited to the allowed range.

Step 4: Memory updating. Update $\vec{p}_{i,\text{best}}$ and $\vec{g}_{i,best}$ when condition is met.

$$\vec{p}_{i,best} = \vec{p}_i \qquad \text{if } f(\vec{p}_i) > f(\vec{p}_{i,best})$$

$$\vec{g}_{i,best} = \vec{g}_i \qquad \text{if } f(\vec{g}_i) > f(\vec{g}_{i,best})$$

where $f(\vec{x})$ is the objective function subject to maximization.

Step 5: Termination Checking. The algorithm repeats Steps 2 to 4 until certain termination conditions are met, such as a pre-defined number of iterations or a failure to make progress for a certain number of iterations. Once terminated, the algorithm reports the values of $\vec{g}_{best}$ and $f(\vec{g}_{best})$ as its solution.

SYSTEM MODELLING RESULTS

Simulations were conducted on a Pentium 4, 3.2GHz computer, in the MATLAB 7 environment. Two systems were used for the identification task, the first given by:

$$G(s) = \frac{Ke^{-T_d s}}{T_2 s^2 + T_1 s + 1} = \frac{7.6e^{-25s}}{10s^2 + s + 1}$$

where the algorithms had to identify K, T_d and T_2 only. The second system was represented by:

$$G(s) = \frac{5.7e^{-42s}}{40.2s^2 + 4.2s + 1}$$

where the algorithms had to identify K, T_d, T_1 and T_2. The proposed parameters were used to create a model output data set which was compared with the training data set (from the known parameters) using the Integral Absolute Error (IAE) as the objective function to be minimised. For some tests, a perturbation has been added to the system response to represent noise (25% of the process signal), since real system measurements are rarely smooth. For both algorithms the population was set to 25 individuals, and a maximum generation of 100. The results of applying the GA and PSO to the identification problem are provided in Table 1. For each parameter the final value determined by the respective algorithm is given followed by its percentage difference from the actual value: for example for the first system, without noise, the PSO determined T_1 as 10.972, 9.72% above the actual value of 10.0. The second to last column presents the number of generations taken to arrive at the determined parameter value, while the final column reports the number of seconds required by the CPU for the complete simulation of 100 generations (determined using the MATLAB function *cputime*). In all cases the PSO computation effort exceeds that of the GA, ranging from 13% to 19% additional time required in comparison.

Table 1 Parameter Identification Variation

	K		T_1		T_2		T_d		No. of Gen.	CPU Time (sec)
	Value	% diff.	value	% diff.	value	% diff.	value	% diff.		
Genetic Algorithm										
No noise	7.600	0.0%	10.006	0.07%	*n/a*	*n/a*	24.992	-0.03%	47	306.4
With noise	7.602	0.03%	10.330	3.3%	*n/a*	*n/a*	25.372	1.5%	46	308.7
No noise	5.700	0.0%	43.380	7.9%	4.379	4.2%	41.072	-2.2%	100	306.0
With noise	5.706	0.1%	41.218	2.5%	4.313	2.7%	41.856	-0.3%	40	314.9
Particle Swarm Optimisation										
No noise	7.945	4.5%	10.972	9.7%	*n/a*	*n/a*	25.109	0.4%	54	365.3
With noise	7.808	2.7%	10.831	8.3%	*n/a*	*n/a*	24.948	-0.2%	56	365.7
No noise	5.668	-0.6%	44.102	9.7%	3.840	-8.6%	42.860	2.1%	89	353.5
With noise	5.575	-2.2%	25.536	-36.5%	3.158	-24.8%	43.389	3.3%	100	356.5

The typical change in Objective Function value over the generations can be seen in Figure 1, illustrating the searching and optimisation processes undertaken by both algorithms.

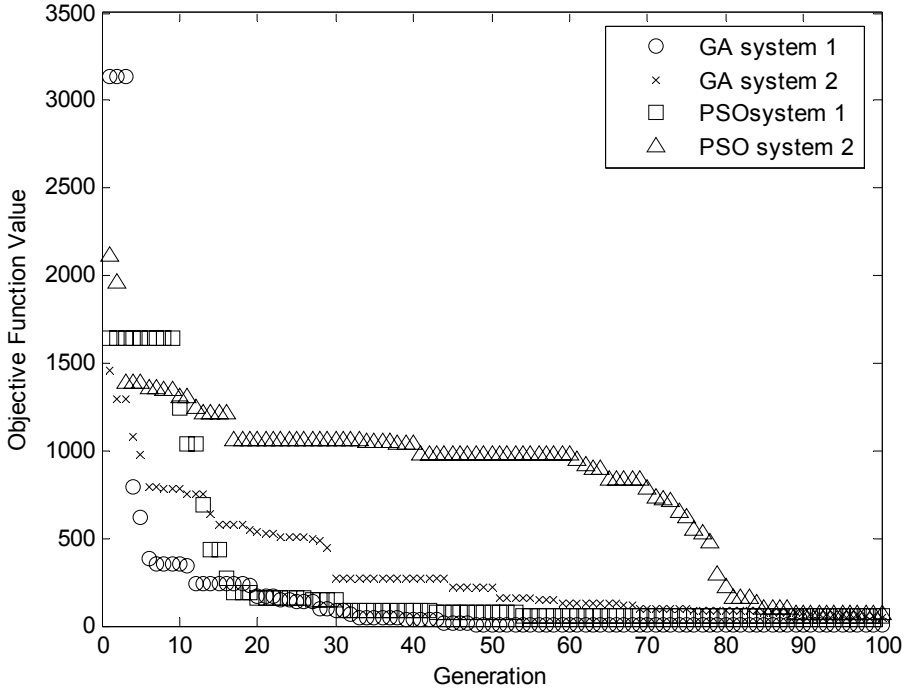

Figure 1 Fitness function over generations.

An indication of the determined model responses from both algorithms are provided in the following figures. Figure 2 (the first system) illustrates that the parameters determined using data corrupt with noise, produce responses that are extremely close matches to the ideal response, with the PSO producing an output which appears closer to the ideal than the GA. A set of results for the second system are shown in Figure 3, where the response from the GA parameters is close to the ideal, however the PSO has developed a model with a significant observable difference from the ideal.

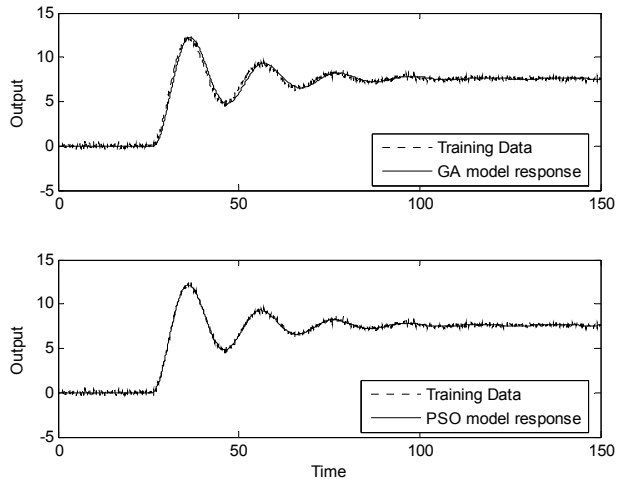

Figure 2 Results for system 1
(K=7.6, T_d=25, T_1 and T_2=10).

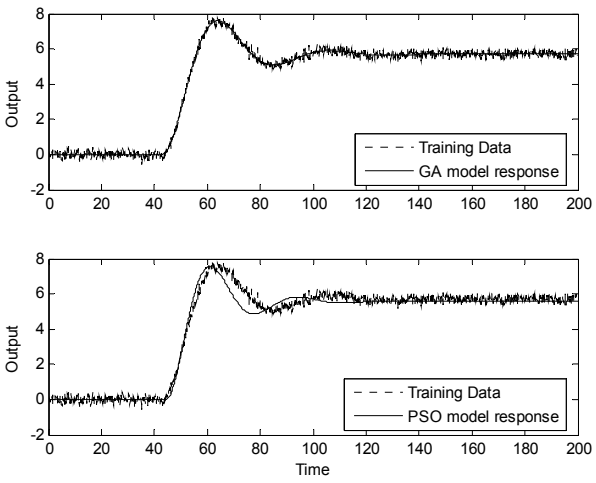

Figure 3 Results for system 2
(K=5.7, T_d=42, T_1=40.2, T_2=4.2).

DISCUSSION

The strength of GAs is in the parallel nature of their search. A GA implements a powerful form of hill climbing that preserves multiple solutions, eradicates unpromising solutions, and provides reasonable solutions. Through genetic operators, even weak solutions may continue to be part of the makeup of future candidate solutions. The genetic operators used are central to the success of the search. All GAs require some form of recombination, as this allows the creation of new solutions that have, by virtue of their parent's success, a higher probability of exhibiting a good performance. In practice, crossover is the principal genetic operator, whereas mutation is used much less frequently. Crossover attempts to preserve the beneficial aspects of candidate solutions and to eliminate undesirable components, while the random nature of mutation is probably more likely to degrade a strong candidate solution than to improve it. Another source of the algorithm's power is the implicit parallelism inherent in the evolutionary metaphor. By restricting the reproduction of weak candidates, GAs eliminate not only that solution but also all of its descendants. This tends to make the algorithm likely to converge towards high quality solutions within a few generations.

Particle Swarm Optimisation shares many similarities with evolutionary computation (EC) techniques in general and GAs in particular. All three techniques begin with a group of a randomly generated population, all utilise a fitness value to evaluate the population. They all update the population and search for the optimum with random techniques. A large inertia weight facilitates global exploration (search in new areas), while a small one tends to assist local exploration. The main difference between the PSO approach compared to EC and GA, is that PSO does not have genetic operators such as crossover and mutation. Particles update themselves with the internal velocity, they also have a memory that is important to the algorithm. Compared with EC algorithms (such as evolutionary programming, evolutionary strategy and genetic programming), the information sharing mechanism in PSO is significantly different. In EC approaches, chromosomes share information with each other, thus the whole population moves like one group towards an optimal area. In PSO, only the 'best' particle gives out the information to others. It is a one-way information sharing mechanism, the evolution only looks for the best solution. Compared with ECs, all the particles tend to converge to the best solution quickly even in the local version in most cases. Compared to GAs, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust.

CONCLUSIONS

Overall the results indicate that both GAs and PSO can be used in the optimisation of parameters during model identification. In terms of computational effort, the GA approach is faster, although it should be noted that neither algorithm takes what can be considered an unacceptably long time to determine their results. With respect to accuracy of model parameters, the GA determines values which are closer to the known values than does the PSO. Finally, the GA seems to arrive at its final parameter values in fewer generations than the PSO. Thus it must be concluded that for the process of process modelling, the GA approach is superior to the PSO approach.

Techniques such as PSO and Genetic Algorithms are inspired by nature, and have proved themselves to be effective solutions to optimization problems. However, these techniques are not a panacea, despite their apparent robustness. There are control parameters involved in these meta-heuristics, and appropriate setting of these parameters is a key point for success. In general, some form of trial-and-error tuning is necessary for each particular instance of optimization problem. Additionally, any meta-heuristic should not be thought of in isolation: the possibility of utilising hybrid approaches should be considered. Additionally for both approaches the major issue in implementation lies in the selection of an appropriate objective function.

REFERENCES

[1] Holland, J. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press.
[2] Negnevitsky, M. 2002. Artificial Intelligence. Addison Wesley.
[3] Pei, M., E. Goodman, & W. Punch. 1997. Pattern discovery from data using genetic algorithms. Proceedings of 1st Pacific-Asia Conference on Knowledge Discovery & Data Mining.
[4] Chen , J.R., R. Nambiar and P. Mars. 1997. Learning Algorithms: Theory and Applications in Signal Processing, Control and Communications. Electronic Engineering Systems. CRC Press.
[5] Van Rooij, A. J. F., L. C. Jain, and R. P. Johnson. 1997. Neural Network Training Using Genetic Algorithms. Machine Perception & Artificial Intelligence. World Scientific Publisher.
[6] Heppener, F. and U. Grenander. 1990. A stochastic nonlinear model for coordinate bird flocks. In Krasner, S. (Ed.) *The Ubiquity of Chaos*. AAAS Publications, USA.
[7] Reynolds, C.W. 1987. Flocks, herds, and schools: a distributed behavioural model. Computer Graphics, 21(4), p. 25-34.
[8] Kennedy, J. and R.C. Eberhart. 2001. Swarm Intelligence. Morgan Kaufman Publishers.
[9] Kennedy, J. and R.C. Eberhart. 1995. Particle Swarm Optimisation. Proceedings IEEE International Conference on Neural Networks, IV, p. 1942-1948.
[10] Eberhart, R.C., Simpson, P. and Dobbins, R. 1996. Computational Intelligence PC Tools. Academic Press.
[11] Koza, J. R. 1991. Genetic evolution and co-evolution of computer programs. In Langton, C. G. C. Taylor, J. D. Farmer and S. Rasmussen (Eds) *Artificial Life II: SFI Studies in the Sciences of Complexity*. **10**. Addison-Wesley.

ABOUT THE AUTHOR

Dr. Karl O. Jones, School of Engineering, Liverpool John Moores University.
Phone: +44 151 231 2199, E-mail: k.o.jones@livjm.ac.uk.