RESEARCH ARTICLE
doi: 10.2306/scienceasia1513-1874.2014.40S.078

ScienceAsia 40S (2014): **78–86**

A combined compact genetic algorithm and local search method for optimizing the ARMA(1,1) model of a likelihood estimator

Rawaa Dawoud Al-Dabbagh[a,*], **Azeddien Kinsheel**[b], **Mohd Sapiyan Baba**[c], **Saad Mekhilef**[d]

[a] Department of Artificial Intelligence, University of Malaya, Kuala Lumpur, Malaysia
[b] Department of Mechanical and Industrial Engineering, Faculty of Engineering, University of Tripoli, Libya
[c] Gulf University of Science and Technology, Kuwait
[d] Power Electronics and Renewable Energy Research Laboratory (PEARL), Department of Electrical Engineering, University of Malaya, Malaysia

*Corresponding author, e-mail: rawaa_aldabbagh@siswa.um.edu.my

Received 13 Feb 2014
Accepted 9 Jun 2014

ABSTRACT: In this paper, a compact genetic algorithm (CGA) is enhanced by integrating its selection strategy with a steepest descent algorithm (SDA) as a local search method to give I-CGA-SDA. This system is an attempt to avoid the large CPU time and computational complexity of the standard genetic algorithm. Here, CGA dramatically reduces the number of bits required to store the population and has a faster convergence. Consequently, this integrated system is used to optimize the maximum likelihood function $\ln L(\phi_1, \theta_1)$ of the mixed model. Simulation results based on MSE were compared with those obtained from the SDA and showed that the hybrid genetic algorithm (HGA) and I-CGA-SDA can give a good estimator of (ϕ_1, θ_1) for the ARMA(1,1) model. Another comparison has been conducted to show that the I-CGA-SDA has fewer function evaluations, minimum search space percentage, faster convergence speed and has a higher optimal precision than that of the HGA.

KEYWORDS: moving average, maximum likelihood, moment estimation, steepest descent algorithm

INTRODUCTION

One of the most famous procedures for the solution of optimization problems is the genetic algorithm (GA), which is composed mainly of three steps: recombination, crossover and mutation. By maintaining a population of solutions, GA can be viewed as an implicit model of the solutions in the search space. In the standard GA, new solutions are generated by applying random recombination operators on two or more high-quality individuals of the current population[1]. These recombination operators, such as one-point, two-point or uniform crossover, randomly selected non-overlapping subsets of two 'parent' solutions to form 'children' solutions.

The poor behaviour of genetic algorithms in some problems, which are sometimes attributed to designed operators, has led to the development of other types of algorithms such as the probabilistic model building genetic algorithm or estimation of distribution algorithm (EDA). There is a class of algorithms which have been developed recently to preserve the building blocks[2]. The principal concept in these new techniques is to prevent the distribution of partial solutions to be included in a solution by building a probabilistic model[2–4]. To name just a few, instances of EDA algorithms include the population-based incremental learning (PBIL)[5,6] and the compact genetic algorithm (CGA)[7]. CGA represents the population as a probability (distribution) vector (PV) over the set of solutions and is operationally equivalent to the order-one behaviour of the simple GA with uniform crossover. It processes each gene independently and requires less memory than simple GA[2,7]. As a second case study to investigate the relative performance of CGA for optimizing the solution of estimation problems, we have integrated CGA with the steepest descent algorithm (SDA) into one system, called I-CGA-SDA, as an attempt to optimize the maximum likelihood function $\ln L(\phi_1, \theta_1)$ of the mixed ARMA(1,1) model.

A time series is an ordered sequence of observations in an equal interval space; this ordering is generated through time or other dimensions such as space. Time series occur in a variety of fields, such as engineering, economics and agriculture. Formally, this series is represented in a stochastic model known as mixed auto-regressive moving average model,

ARMA(1,1)[8],

$$z_t = \phi_1 z_{t-1} + \phi_p z_{t-p} + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q} \quad (1)$$

or

$$\phi(B) z_t = \theta(B) a_t \quad (2)$$

where $a_t \sim \text{iid } \mathrm{N}(0, \sigma_a^2)$, which means that the a_t's are identically, independently distributed, each with a normal distribution having mean 0 and the same variance. This model employs $p + q + 2$ unknown parameters, μ, ϕ_1, $\phi_2, \ldots, \phi_p$, θ_1, $\theta_2, \ldots, \theta_q$, σ_a^2, that are estimated from the data. The moving average process is stationary for any value of θ_1, $\theta_2, \ldots, \theta_q$, that is, the mean and the variance of the underlying process are constant and the auto-covariance depends only on the time lag. But many economic and business time series are sometimes considered to be non-stationary. Non-stationary time series can occur in many different ways. Sometimes the series have a non-stationary behaviour about a fixed mean, and hence its behaviour can be represented by a model which calls for dth difference of the process to be stationary. In practice d is usually 0, 1, or at most 2. This behaviour can be represented by (p, d, q) model for the dth difference to be stationary[8]. Then the model takes the formula

$$w_t = \phi_1 w_{t-1} + \cdots + \phi_p w_{t-p} + a_t - \theta_1 a_{t-1}, \quad (3)$$

or, in its operator form,

$$\phi(B) w_t = \theta(B) a_t, \quad (4)$$

where $w_t = \Delta^d z_t$. With the moving average operator, (4) can be represented by

$$\phi(B)(1 - B)^d z_t = \theta(B) a_t, \quad (5)$$

which provides a powerful model for describing stationary and non-stationary time series, and it is called an integrated moving average model (IMA) process or ARIMA of order $(0, d, q)$. A mixed model of first-order is known as first-order auto-regressive moving average model, which is denoted by ARMA(1,1). Then (1) and (2) reduces to

$$z_t = \phi_1 z_{t-1} + a_t - \theta_1 a_{t-1}, \quad (6)$$

or

$$(1 - \phi_1 B) z_t = (1 - \theta_1 B) a_t. \quad (7)$$

This model will be invertible and stationary if all the roots of $(1 - \phi_1 B)$ lie outside the unit circle. An invertible MA is time-reversible, so we can get that $|\phi_1| < 1$ and $|\theta_1| < 1$.

Estimation is the second step in analysis of the time series. It indicates an efficient use of the data to make inferences about the parameters conditional to the adequacy of the entrained model. ARMA models can be difficult to estimate if the parameter estimates are not within the appropriate range, a moving average model's residual terms will grow exponentially. The calculated residuals for later observations can be very large or can overflow. This can happen either because of improper starting values being used or because the iterations moved away from reasonable values. Moreover, our model is nonlinear because $a_t = ((1 - \phi_1 B)/(1 - \theta_1 B)) z_t$, so there is no direct method that can handle these limitations, but all suitable methods are indirect methods (iterative methods) which start with an initial value, then this value is iteratively modified by using some numerical algorithms. The numerical method gives an approximate estimator with some accuracy.

Recently, Hussain[9,10] proposed the use of a canonical genetic algorithm for optimizing the maximum likelihood function $\ln L(\theta, \sigma_a^2)$ of the first-order moving average MA(1) model. The results were compared with the results obtained by the moment estimator method. A hybrid GA and steepest descent method was then proposed to optimize the likelihood estimator of ARMA(1,1) model, and the results were quite encouraging, compared to those from CGA[11].

In this paper, we introduce a new evolutionary way to estimate the same model by using CGA integrated with steepest descent method as a local search. In literature, Droste[12] proved that CGA is applicable for optimizing most of the linear functions such One-Max in the optimal expected runtime. In this paper we are interested in the behaviour of the CGA for nonlinear functions in comparison with CGA according to the number of function evaluations taken, solution quality, the percentage of the search space searched until convergence and the convergence speed. Moreover, Al-Dabbagh et al used CGA to optimize the maximum likelihood estimator of the first-order moving average model MA(1)[13]. Simulation results based on MSE were compared with those obtained from the moment's method and showed that CGA can give good estimator of the MA(1) model. Another comparison has been conducted to show that the CGA method has fewer function evaluations, minimum searched space percentage, faster convergence speed and has a higher optimal precision than that of the CGA.

MATHEMATICAL FORMULATION

Maximum likelihood estimator (MLE) is a standard approach to parameter estimation and inference in statistics; it is a method that finds the most likely

value for the parameter based on the data set collected, in particular in nonlinear modelling with non-normal data. MLE has many optimal properties in estimation: sufficiency (complete information about the parameter of interest contained in its MLE estimator); consistency (true parameter value that generated the data recovered asymptotically, i.e., for data of sufficiently large samples); efficiency (lowest-possible variance of parameter estimates achieved asymptotically); and parameter invariance (same MLE solution obtained independent of the parameter used)[14].

In order to study the ARMA(1,1) model, let us assume that a time series which is denoted by $z_{-d+1}, \ldots, z_0, z_1, z_2, \ldots, z_n$ is generated by (6) modelled over $N = n + d$ original observations z. Then, the stationary mixed ARMA(0,1) model in (6) can be rewritten as[14,15],

$$a_t = w_t + \phi_1 w_{t-1} + \theta_1 a_{t-1} \tag{8}$$

where $E(w_t) = 0$. Suppose that $\{a_t\}$ has a normal distribution with zero mean and constant variance equal to σ_a^2, then the likelihood function can be written as follows[14,15]:

$$L = \sqrt{\frac{\left|M^{(1,1)}\right|}{(2\pi\sigma_a^2)^n}} \exp\left(\frac{-s(\phi_1,\theta_1)}{2\sigma_a^2}\right) \tag{9}$$

where

$$\begin{aligned} M^{(1,1)} &= \text{var-cov}(\phi_1,\theta_1) \\ &= I^{-1}(\phi_1,\theta_1)\frac{1}{I(\phi_1,\theta_1)}\text{adj}(I(\phi_1,\theta_1)). \end{aligned} \tag{10}$$

If $M^{(1,1)} = I^{-1}(\phi_1,\theta_1)$, then the logarithmic likelihood function will be given by,

$$\ln L = -\frac{n}{2}\ln(2\pi\sigma_a^2) + \frac{1}{2}\ln\left|M^{(1,1)}\right| - \frac{s(\phi_1,\theta_1)}{2\sigma_a^2}, \tag{11}$$

where

$$I(\phi_1,\theta_1) = \frac{n}{\sigma_a^2}\begin{bmatrix} \frac{\sigma_a^2}{1-\phi^2} & \frac{\sigma_a^2}{1-\phi_1\theta_1} \\ \frac{\sigma_a^2}{1-\phi_1\theta_1} & \frac{\sigma_a^2}{1-\theta^2} \end{bmatrix} \tag{12}$$

and

$$s(\phi_1,\theta_1) = \sum_{t=-\infty}^{n} (a_t \mid \phi_1,\theta_1,w)^2 \tag{13}$$

is the sum of squared errors, while $(a_t \mid \theta_1,\phi_1,w)$ denotes the conditional expectation of a_t given θ_1,ϕ_1 and w. The sum of squared errors can be found by unconditional calculation of the a_t's, which is computed recursively by taking expectations in (13), it is also called *Least Square Estimate* in which the parameter estimated is obtained by minimizing the sum of squares in (13), it usually provides very close approximation to the maximum likelihood estimator. Back-forecasting is a popular technique, it estimates the parameters which are crudely put into the model and run backwards in time. A back-calculation provides the values w_{-j} for $j = 0, 1, 2, \ldots$, which is needed to start off the forward recursion. For moderate and large values of n, Equation (13) is dominated by $s(\phi_1,\theta_1)/2\sigma_a^2$, and thus the contours of the unconditional sum of squares function in the space of the parameters (ϕ_1,θ_1) are very much nearly contours of likelihood and log-likelihood.

INTEGRATED COMPACT GENETIC ALGORITHM AND STEEPEST DESCENT ALGORITHM (I-CGA-SDA)

Compact genetic algorithm

The CGA is drawn from the PBIL, but requires fewer steps, fewer parameters and fewer gene samples[12]. The CGA manages its population as the probability vector PV over the set of solutions (i.e., only models its existence), thereby mimicking the order-one behaviour of the SDA with uniform crossover using a small amount of memory[1,16].

Fig. 1 describes the pseudo-code of the CGA. The values of PV are constrained by $p_i \in [0,1]$ for all $i = 1, \ldots, l$, where l is the number of genes (i.e., the length of the chromosome), which measures the proportion of '1' alleles in the ith locus of the simulated population[2,7]. The PV is initially assigned the values of 0.5 to represent a randomly generated population. In every generation (i.e., iteration), competing chromosomes are generated on the basis of the current PV, and their probabilities are updated to favour a better chromosome (i.e, winner). It is noted that the generation of chromosomes from PV simulates the effects of crossover that leads to a decorrelation of the population's genes. In a simulated population of size n, the probability p_i is increased (or decreased) by $1/n$ when the ith locus of the winner has an allele of '0' (or '1'). If both the winner and the loser have the same allele in each locus, then the probability remains the same. This scheme is equivalent to (steady-state) pairwise tournament selection. The CGA is terminated when all the probabilities converge to zero or one. The convergent PV itself represents the final solution. It can be seen that the CGA requires $l \times \log_2(n+1)$ bits of memory while the SDA requires $l \times n$ bits[1]. Thus a large-size population can be effectively exploited

Parameters. n: population size, l: chromosome length
Step 1. Initialize probability vector
  for $i := 1$ to l **do** $p_i := 0.5$
Step 2. Generate two chromosomes from the probability vector
  $a :=$ **generate**(p); $b :=$ **generate**(p)
Step 3. Let them compete
  winner, loser := **compete**(a, b)
Step 4. Update the probability vector
  for $i := 1$ to l **do**
    if winner(i) **then**
      $p_i := p_i + 1/n$
    else if loser(i) **then**
      $p_i := p_i - 1/n$
    end if
  end for
Step 5. Check if the probability vector has converged. Go to **Step 2**, if it is not satisfied.
Step 6. The probability vector p_i represents the final solution.

Fig. 1 Pseudo-code fashion of the CGA.

without unduly compromising on memory requirements [16, 17].

(1) Individual Initialization and encoding: Certain restrictions are defined on the encoding scheme:

(i) CGA needs in every step two random numbers, each having a bit-string (0's and 1's) of fixed length $l = 15$ bits. Two individuals a and b are generated: they are two identical chromosomes working in parallel, but using different initial seeds. Each individual affectionately known as a critter represents an element with the domain of the solution space of the optimization problem. The chromosome of a given critter is the only source for of all the information about the corresponding solution. To apply the CGA to real-values parameters optimization problems of the form $f : \prod[u_i, v_i] \rightarrow R(u_i < v_i)$, the bit-strings is logically divided into n segments of (in most cases) equal length l_x $(l = nl_x)$ and each segment is interpreted as the binary code of the corresponding object variable $x_i \in [u_i, v_i]$. A segment decoding function $\Gamma^i : \{0, 1\}^{l_x} \rightarrow [u_i, v_i]$ typically looks like,

$$\Gamma^i(a_{i1}a_{i2}\cdots a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1}\left(\sum a_{ij}2^{j-1}\right),$$

where $(a_{i1}a_{i2}\cdots a_{il_x})$ denotes the ith-segment of an individual $\mathbf{a} = (a_{11}, \ldots, a_{nl_x}) \in l^{nl_x} = I^l$ (Fig. 2).

(ii) Every field of the probability vector PV is initialized to 0.5.

(2) Fitness Evaluation: A fitness function is a numerical value associated with each individual to measure the goodness of the solution. Each individual a and b is converted into a number between 0 and 32 768 (15 bits mean 2^{15} possible values). Hence individual with higher fitness value represents better solution, while lower fitness value is attributed to the individual whose bit-string represents inferior solution. Combining the segment-wise decoding function to individual-decoding function $\Gamma = \Gamma^1 \times \cdots \times \Gamma^n$ [18, 19], fitness values are obtained by setting

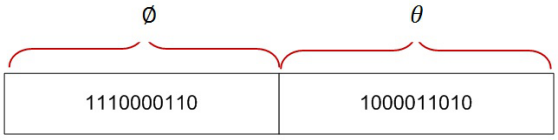

Fig. 2 The individual structure in CGA.

$$\phi(\mathbf{a}) = \delta\left(f(\Gamma(\mathbf{a}))\right)$$

where δ denotes a scaling function ensuring positive fitness values such that the best individual receives largest fitness.

(3) Compete: Compete is a procedure that compares two real-values (meaning 2 bit-strings), a and b and has an output either '1' (if $a > b$), or '0' (if $a < b$). The comparison depends on the Fitness Evaluation module.

(4) Probability Update: As the population has n chromosomes, the probability vector PV must be able to be increased or decreased by a minimal value of $1/n$. There is no need to represent the probability as the float number is.

As the probability always has a value between 0 and 1, and can be written as the sum of the negative powers of 2, with 0 and 1 as coefficients, the probability vector contains the bit-string of these coefficients. Increasing and decreasing it by the minimal value means to change at least one value of this bit-string. Technically, the p_i is updated as follows:

If $f_a \geqslant f_b$ then
  if $a_i = 1$ then $p_i = \min\left(1, p_i + \frac{1}{n}\right)$
  if $a_i = 0$ then $p_i = \max\left(1, p_i - \frac{1}{n}\right)$
else
  if $b_i = 1$ then $p_i = \min\left(1, p_i + \frac{1}{n}\right)$
  if $b_i = 0$ then $p_i = \max\left(1, p_i - \frac{1}{n}\right)$.

Hence p_i's store the bit-string that represents the probability. The operations that it needs are increment and decrement of the bit-string by one unit.

Local search: steepest descent method

Typically, a local search method searches for the best solution starting at a previously selected point, which in our case is a solution selected throughout the evolution process of CGA. In this paper, we use steepest descent method to play the role of the local search operator. It is defined by three basic stages: frequency, probability, and the number of local search iterations.

(1) Steepest Descent Method: Steepest Descent Algorithm (SDA) [19] is an iterative algorithm that depends on the following rules of numerical computations:

$$\beta_i^* = \beta_{i-1} - k\nabla\, \mathrm{e}^2$$

where β_{i-1} is the parameter model, k is a constant value (to be derived later), $\nabla\, \mathrm{e}^2$ is the gradient of e^2, which can be approximated by $\nabla\, \mathrm{e}^2 = (\partial\, \mathrm{e}^2/\partial\beta_1, \partial\, \mathrm{e}^2/\partial\beta_2, \ldots, \partial\, \mathrm{e}^2/\partial\beta_m)$. We can see that the estimation of the parameters depends on an iterative method which starts with an initial value β_i (estimated form one of the traditional estimation methods). This algorithm will continue in modifying these estimators until it reached to a stage that there will be no change in the values of the forecasting mean square error (FMSE), which is given by,

$$\mathrm{FMSE} = \frac{1}{n-1}\sum_{t=1}^{n}(z_t - \hat{z}_t)^2$$

where z_t is an actual value of observed time series and $\hat{z}_t$ is a predicted value of the actual one. From (6), we obtain

$$a_t^2 = (z_t - \phi_{1t}w_{t-1} + \theta_{1t}a_{t-1})^2.$$

Therefore, $\partial a_t^2/\partial\phi_{1t} = -2a_t z_{t-1}$ and $\partial a_t^2/\partial\phi_{1t} = -2a_t a_{t-1}$. So,

$$(\phi_{1t}^*, \theta_{1t}^*) = (\phi_{1t} + 2ka_t z_{t-1}, \theta_{1t} - 2ka_t a_{t-1}).$$

In order to obtain the value of k, we first note that

$$|\Delta a_t| = |a_t^* - a_t| = 2ka_t\left(z_{t-1}^2 + a_{t-1}^2\right)$$

Since $0 < |\Delta a_t/a_t| < 1$, we obtain

$$0 < k < \frac{1}{2\left(z_{t-1}^2 + a_{t-1}^2\right)}.$$

As a local search method, this algorithm moves along the direction of the steepest gradient until an improved point achieved. The algorithm ends when there is no new relationship shown point can be found (i.e., when the gradient equal zero).

(2) The number of Local Search Iterations: One important issue for the application this algorithm is how long the local search lasts before switching back to the global CGA search. In order to make this decision, we compare the most recent fitness improvements by local search with last fitness improvements by global search.

$$\text{Do Local Search if } \frac{\Delta\mathrm{Global}}{\mathrm{pop}} < \frac{\Delta\mathrm{Local}}{\mathrm{fev}}. \quad (14)$$

This criterion presented in (14) where is the improvement achieved between the two previous global search generations, $\Delta\mathrm{Local}$ is the current improvement in the local search step, pop is the population size, and fev is the number of function evaluations required for the local search step. This criterion scales the fitness improvements by the computational effort required (i.e., pop and fev) so that the ratios are comparable. When (14) is no longer true, or when the number of iterations exceeds a user-specified maximum value, the algorithm switches back to the global search.

Conceptual integrated algorithm

The coupling approaches are applied by the introduction of the generation interval for Hybrid Activation Operator (HAO). When CGA goes through its operators generation by generation, the HAO is activated since CGA achieves 20 generations. The intermediate generation created by CGA is fed into an adopted selection strategy which selects the best solutions obtained from all previous generations into an array of a small size. Then each binary individual in this array is converted into real-value to be the initial values of steepest descent algorithm that operates on this array

Parameters. n_p: population size, l: chromosome length
Step 1. Initialize probability vector
for $i := 1$ to l **do** $p[i] := 0.5$
Step 2. Generate two chromosomes from the probability vector
$a :=$ **generate**(p); b:= **generate**(p)
Step 3. Let them compete
winner, loser := **compete**(a, b)
Step 4. Update the probability vector
for $i := 1$ to l **do**
if winner$[i]$ **then**
$p[i] := p[i] + 1/n$
else if loser$[i]$ **then**
$p_: = p_i - 1/n$
end if
end for
Step 5. Check if the probability vector has converged.
Go to **Step 2** if it is not satisfied, otherwise go to **Step 7**.
Step 6. Local Search implementation
Check if the number of iterations has achieved the desired for applying the local search.
Step 6.1 Convert all the binary vectors in into their corresponding real-values.
Step 6.2 Apply SDA.
Step 6.3 Re-convert the best solution achieved to its corresponding binary value, go to **Step 2**.
Step 7. The probability vector $p[i]$ represents the final solution.

Fig. 3 Pseudo-code fashion of I-CGA-SDA system.

for fixed small number of iterations. Then the best vector among all the obtained vectors is converted back into binary string to be manipulated by CGA. Here, steepest descent method used as a tool in this integration system that operates in small number of generations in order to enhance the selected points driven from CGA. The steps of the new proposed I-CGA-SDA algorithm are stated in Fig. 3.

SIMULATION RESULTS AND DISCUSSION

This section presents the simulation results obtained and the comparison conducted between the I-CGA-SDA and HGA in terms of solution quality, the number of function evaluations and the percentage of the searched space taken for the likelihood estimator of ARMA(1,1)[10,11]. All simulation results are attained by triggering 5 distinct runs; each run has 100 generations, then averaging the results data. Furthermore, the results of these former methods have been compared with those obtained by steepest descent method based on initial values obtained by moment method for the same value of (ϕ, θ) with 1000 runs. The simulation results performed are based on different sample size (i.e., $n = 25, 75, 125$), ϕ is set to ($\pm$0.1, $\pm$0.3, $\pm$0.4, $\pm$0.6), and θ is set to ($\pm$0.2, $\pm$0.4, $\pm$0.5, $\pm$0.8). The random variables a_t's are generated by using Box-Muller formula and sample of size n generated by (3). The comparison has been based on Mean Square Error, $\mathrm{MSE} = \mathrm{var}(\theta) + \mathrm{bias}$.

The HGA used binary tournament selection without replacement, and uniform crossover with exchange probability $P_\mathrm{c} = 0.75$. Inversion mutation is used with probability $P_\mathrm{m} = 0.005$. The population size is set to 50. All runs end when the population fully converged that is when the individuals have the same alleles at each gene position.

As opposed to HGA, in I-CGA-SDA the population size P_s and the chromosome length l are set to 30–50 and 20, respectively. The algorithms start with a probability register initialized with 0.5, so that at the beginning, there are equal chances for every bit of the future chromosome to be either '0' or '1' at the end of the algorithm. The objective function decides whether it is better to increase or decrease the entry in the probability register. Table 1 illustrates the results and the simulations on a set of data that gives some ideas of the behaviour of HGA, I-CGA-SDA, and SDA.

From Table 1 we can see that the MSE of HGA and I-CGA-SDA are relatively competing in a small range of differences but they are all smaller than those obtained from the SDA. Consequently, they are more reliable than the SDA in estimating the parameters of the model under study. On the other hand, the value of the MSE decreases when the sample size increases for all the adopted methods. Moreover, MSE of I-CGA-SDA and HGA when the model parameters (ϕ_1, θ_1) take positive values is smaller than that when these parameters are assigned to negative ones.

Table 1 MSE for HGA, I-CGA-SDA, and SDA methods for different values of sample size and model parameter.

n	ϕ	θ	SDA		HGA		I- CGA-SDA	
			ϕ	θ	ϕ	θ	ϕ	θ
25	0.6	0.8	1.37	1.52	0.56	0.782	0.32	0.686
	0.4	0.5	0.83	1.1	0.601	0.602	0.36	0.506
	−0.1	0.2	0.58	0.38	0.28	0.243	0.21	0.225
	−0.3	−0.4	0.64	1.04	0.48	0.521	0.271	0.511
75	0.6	0.8	1.36	1.43	0.503	0.657	0.271	0.563
	0.4	0.5	0.8	0.97	0.53	0.482	0.346	0.425
	−0.1	0.2	0.3	0.24	0.248	0.241	0.143	0.181
	−0.3	−0.4	0.61	0.801	0.472	0.416	0.248	0.406
125	0.6	0.8	1.32	1.41	0.414	0.431	0.226	0.551
	0.4	0.5	0.75	0.96	0.275	0.431	0.201	0.221
	−0.1	0.2	0.36	0.212	0.226	0.225	0.122	0.025
	−0.3	−0.4	0.52	0.61	0.216	0.41	0.112	0.301

Table 2 illustrates the average simulation results of HGA and I-CGA-SDA, respectively, with population size $P_{\mathrm{s}} = 50$ over 100 runs, where F is the number of function evaluations taken until convergence for the various numbers of generations, and PSS is the percentage of the searched space which can be calculated as follows[13]:

$$\mathrm{PSS} = \frac{\mathrm{NC} \times \mathrm{NG}}{\mathrm{TSS}}$$

where NC = number of individuals being evaluated per generation, NG = number of generations until convergence, TSS = total search space size. Here, total search size $= 2^l \times$ (no. of chrom. being evaluated), which is equal to P_{s} in HGA and 2 in I-CGA-SDA. Formally speaking, there is an evidence that the two algorithms are quite different, while HGA has a memory requirement of $l \times P_{\mathrm{s}}$, the I-CGA-SDA requires only $l \times \log_2 P_{\mathrm{s}}$ bits and in the number of function evaluations HGA requires $P_{\mathrm{s}} \times \mathrm{NG}$, while I-CGA-SDA requires only $2 \times \mathrm{NG}$. As one can see from the results illustrated in Table 2, the difference between I-CGA-SDA for both the number of function evaluations and the percentage of the searched space until convergence in which I-CGA-SDA exhibits better performance than in the average of both cases. It is also worth noting that the number of function evaluations and the searched space are both decreases when the number of sample size increases.

From Fig. 4, it is clear that the quality of solutions and convergence speed found by the I-CGA-SDA is better than these obtained by HGA. Ultimately, the results suggest that the I-CGA-SDA performs the best and the HGA performs the worst.

Table 2 Average simulation results of HGA and I-CGA-SDA based on the number of function evaluations F and the percentage of the searched space *PSS*.

n	ϕ	θ	HGA		I-CGA-SDA	
			F	PSS	F	PSS
25	0.6	0.8	3200	9.7656	124	0.3784
	0.4	0.5	4150	12.6647	130	0.3967
	−0.1	0.2	3000	9.1552	116	0.354
	−0.3	−0.4	3750	11.444	124	0.3784
75	0.6	0.8	3200	9.7656	122	0.3723
	0.4	0.5	3650	11.1389	126	0.3845
	−0.1	0.2	2800	8.54492	100	0.3051
	−0.3	−0.4	3500	10.6811	120	0.3662
125	0.6	0.8	2250	6.8664	96	0.2929
	0.4	0.5	3000	9.1552	106	0.3234
	−0.1	0.2	2450	7.4768	98	0.299
	−0.3	−0.4	3250	9.9182	116	0.354

System Configuration

Intel Core Duo CPU T6670 @ 2.20 GHz 2.20 GHz, 4.00 GB of memory, Windows 7 Professional Version 2009, Language: Delphi 7.

CONCLUSIONS

In this paper, we investigate the performance of a new integration system I-CGA-SDA for estimating the parameter of log-likelihood function of first order moving average model ARMA(1,1). Based on MSE, I-CGA-SDA provides effective results for three random samples with different sizes (n = 25, 75, 125) with (ϕ, θ) that are set to ($\pm$0.1, $\pm$0.3, $\pm$0.4, $\pm$0.6) and ($\pm$0.2, $\pm$0.4, $\pm$0.5, $\pm$0.8), respectively; in comparison with the HGA and SDA methods. Simulation results also show that the I-CGA-SDA has a higher optimal precision or at least the same as that obtained from the HGA, at same time, the I-CGA-SDA needs minimum searched space percentage and fewer function evaluations than that of the HGA.

Acknowledgements: The authors would like to thank University of Malaya for providing financial support under the UMRG research grant project no. RG111-12ICT.

REFERENCES

1. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.
2. Larrañaga P, Lozano JA (2002) *Estimation of Distribution Algorithms: A New Tool For Evolutionary Computation*, Springer, the Netherlands.

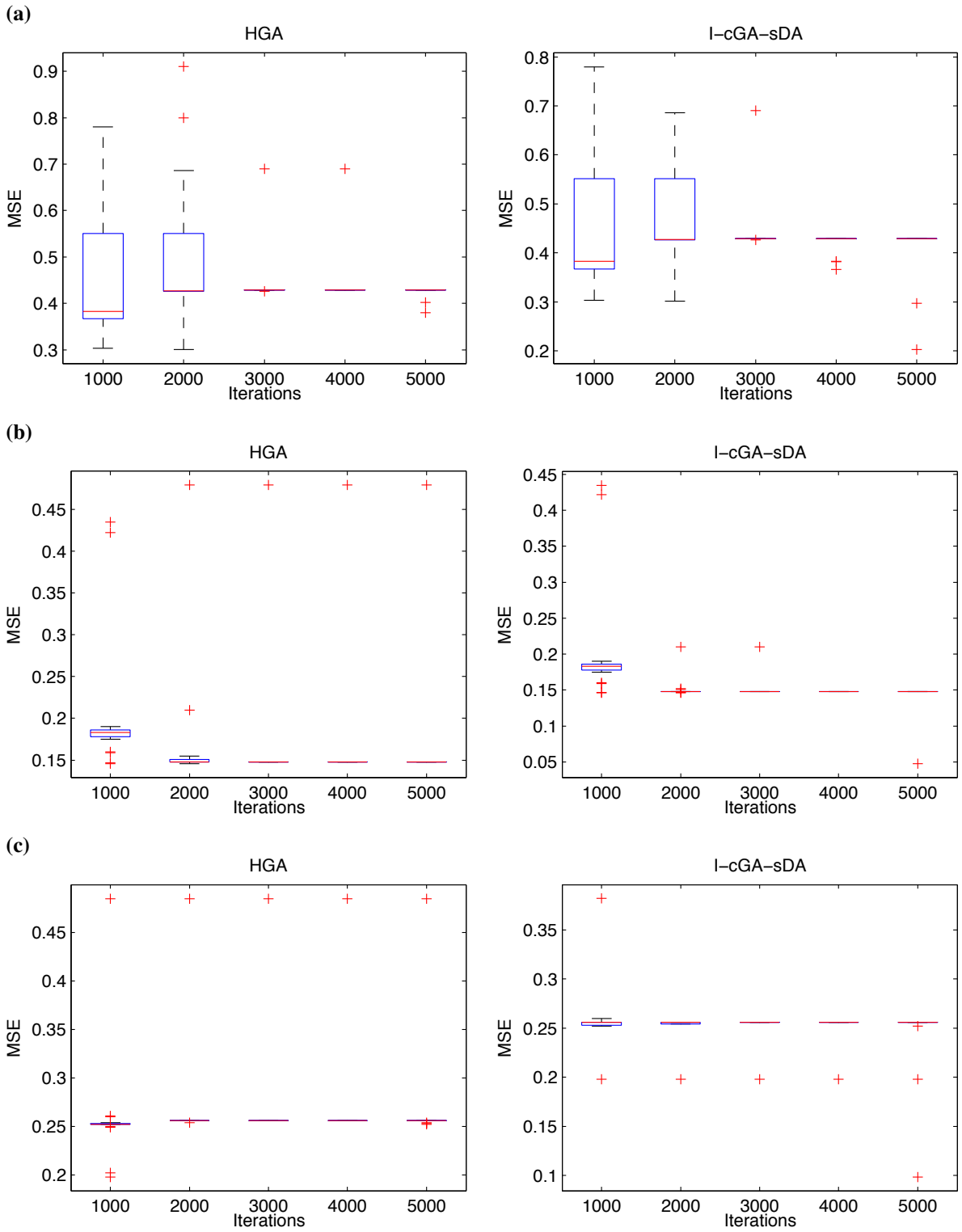

Fig. 4 Convergence graph using box-and-whisker diagrams for HGA and I-CGA-SDA performance for 100 generations over 5 runs (a) $\phi_1 = -0.4$, and $\theta_1 = 0.1$, $n = 25$, (b) $\phi_1 = 0.6$, and $\theta_1 = 0.8$, $n = 75$, (c) $\phi_1 = -0.4$, and $\theta_1 = -0.3$, $n = 125$. The horizontal axis is the number of iterations, and the vertical axis is the median value. The figure also shows the minimum and maximum (+) MSE achieved through iterations.

3. Pelikan M, Goldberg DE, Lobo FG (2002) Survey of optimization by building and using probabilistic models. *Comput Optim Appl* **21**, 5–20.
4. Rastegar R, Hariri A (2006) A step forward in studying the compact genetic algorithm. *Evol Comput* **14**, 277–89.
5. Baluja S (1994) *Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning*, tech. report CMU-CS-94-163, Computer Science Department, Carnegie Mellon Univ, Pittsburgh, PA.
6. Baluja S, Caruana R (1995) Removing the genetics from the standard genetic algorithm. In: Prieditis A (ed) *Proceedings of the Twelfth International Conference on Machine Learning*, pp 38–46.
7. Harik GR (1999) The compact genetic algorithm. *IEEE Trans Evol Comput* **3**, 287–97.
8. Wei WWS (1994) *Time Series Analysis: Univariate and Multivariate Methods*, 2nd edn, Addison-Wesley, Redwood City, CA.
9. Hussain B, Al-Dabbagh RD (2007) A canonical generic algorithm for likelihood estimator of first order moving average model parameter. *Neural Netw World* **17**, 271–82.
10. Al-Sarray BAH, Al-Dabbagah RD (2011) Variants of hybrid genetic algorithms for optimizing likelihood ARMA model function and many of problems. In: Kita E (ed) *Evolutionary Algorithms*, InTech, pp 219–46.
11. Hussain BA, Al-Dabbagh RD (2008) Hybrid canonical genetic algorithm and steepest descent algorithm for optimizing likelihood estimators of ARMA (1,1) model. In: *Proceedings of the First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, pp 30–7.
12. Droste S (2006) A rigorous analysis of the compact genetic algorithm for linear functions. *Nat Comput* **5**, 257–83.
13. Al-Dabbagh RD, Baba MS, Mekhilef S, Kinsheel A (2012) The compact Genetic Algorithm for likelihood estimator of first order moving average model. In: *Proceedings of the Second International Conference on Digital Information and Communication Technology and its Applications (DICTP2012)*, pp 474–81.
14. Box GEP, Jenkins GM (1994) *Time Series Analysis: Forecasting and Control*, Prentice Hall PTR.
15. Myung IJ (2003) Tutorial on maximum likelihood estimation. *J Math Psychol* **47**, 90–100.
16. Aporntewan C, Chongstitvatana P (2011) A hardware implementation of the Compact Genetic Algorithm. In: *Proceedings of the 2001 Congress on Evolutionary Computation*, pp 624–9.
17. Tsutsui S (2002) Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. In: Merelo Guervós JJ, et al (eds) *Parallel Problem Solving from Nature—PPSN VII*. Springer, Berlin, pp 224–33.
18. Makridakis S, Wheelwright SC, Hyndman RJ (1997) *Forecasting: Methods and Applications*. Wiley, New York, NY.
19. Chrysoula DF (2011) Yule-Walker estimation for the moving-average model. *Int J Stoch Anal* **2011**, 151823.