Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Applied Soft Computing 3 (2003) 1–21

Applied Soft Computing
www.elsevier.com/locate/asoc

# An evolutionary approach to the automatic generation of mathematical models

Anjan Kumar Swain[a], Alan S. Morris[b,*]

[a] Indira Gandhi Institute of Technology, Sarang, India
[b] Department of Automatic Control and Systems Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, UK

Received 10 December 2000; received in revised form 9 July 2001; accepted 15 October 2002

**Abstract**

This paper deals with the development and analysis of an efficient, evolutionary, intelligent, data-based, mathematical modelling method for complex non-linear systems. A new hybrid evolutionary method using genetic programming (GP) and evolutionary programming approaches is proposed. The potentials of the hybrid method for modelling complex, dynamic systems including single and two-link terrestrial manipulator systems are investigated and simulation results are presented.
© 2003 Elsevier Science B.V. All rights reserved.

*Keywords:* Genetic programming; Evolutionary programming; Evolutionary computation; Single-link manipulator; Two-link manipulator

## 1. Introduction

The issue of intelligent, data-based modelling, in the context of evolutionary computation (EC), will be the primary focus of this paper. The potential of EC methods for data-based modelling will be investigated and a new EC-based hybrid method will be developed to address the vital issue of data-based modelling. It will be shown that, in general, genetic programming (GP)-based evolutionary methods for data-based modelling provide a clear understanding of the internal structure of any system. While concentrating on the development of accurate systems models, the demerits of approximate system models will be analysed. Further, it will be argued that the results of this paper can be true in general for any data-based modelling techniques. In addition, it will be shown that the automatic generation of mathematical models from the knowledge of the only input–output dataset with the use of evolutionary methods are not suitable for even two-link manipulator systems. Thus, its applicability will only be limited to very simple single-link manipulator systems.

Neural networks are widely used for data-based modelling that is subsequently used for the control of a system [1–4]. However, the use of neural networks for the modelling of even simple robotic manipulators has exhibited very limited potential [1]. In addition, the major problem associated with neural modelling is that its non-transparent internal structure inhibits any theoretical understanding of the model obtained. Hence, the analysis of the global and local behaviour of the underlying model is very difficult with a neural modelling approach. This problem of the non-transparent characteristics of the popular

* Corresponding author. Tel.: +44-114-222-5130; fax: +44-114-222-5661.
*E-mail addresses:* swainanjan@hotmail.com (A.K. Swain), a.morris@sheffield.ac.uk (A.S. Morris).

neural networks can be addressed with a more transparent symbolic approach like GP. The GP and other EC methods are discussed briefly in Section 2.

Traditionally, dynamic system modelling problems are solved with three distinct steps. Initially, a suitable model structure for the given system is selected. Then, the parameters contained in the assumed model are optimised, and subsequently the dataset is validated. Hence, accurate initial structure selection for a given system is extremely important for proper system modelling. But, in theory, an infinite number of models can be built for a given set of data. This necessitates a judicious development of very effective algorithms that can quickly transform the initially selected model into the optimal model of the system. This problem can be stated as:

> For a given input–output dataset, define a $\mu_t$ number of possible initial model structures. Then, find the most appropriate model structure amongst them by manipulating their respective numerical parameters to best fit the given dataset.

The GP-based approaches have the advantage of providing a clear view of the underlying structure of the problem, thus allowing in-depth analysis of the internal structure plasticity during and after learning. This provides a better understanding of the problem. GP-based methods have been used with many reported successes for modelling of moderately complex dynamical systems [5–8]. The inherent structure of the tree-coded genetic programming methods can be used to represent mathematical expressions in modelling simple non-linear systems. Thus, tree-coded GP methods can be used for the automatic generation of mathematical models for manipulator systems. GP works for any problem by randomly selecting initial tree-structured computer programs that can represent initial models of the problem. Further, with the use of variation operators, these structures change to search for better structures. The power of the GP can further be enhanced by changing the numerical ephemeral values associated with each structure. If a particular structure that can exactly model the given dataset is present, this may perform badly due to unsuitable numerical values being used for that structure, and this may cause that structure to be eliminated altogether from the competition. This can be overcome by using a good optimisation technique to optimise the numerical values of each structure along with its own evolution [9]. The major concern here is the associated computational cost, as such a GP is computationally highly intensive. This has been addressed in this work by updating the numerical values of only the best individual structure to improve its fitness value.

Perhaps the first use of hybrid GP and genetic algorithms (GAs) for data modelling was reported by Howard and D'Angelo [10], who used it to finding a mathematical relationship between physical and biological parameters of a stream ecosystem. In an attempt to model robotic manipulators, Castillo and Melin [11] suggested a hybrid fuzzy–fractal–genetic method with comparatively impressive initial results particularly for single-link manipulators. They used a fuzzy–fractal method for modelling and a fuzzy–genetic method for simulation. Unfortunately, their work does not include sufficient results to make further comments on their proposed method. Cao et al. [9] used GA to optimise the parameters of the tree-structured GP individuals to preserve useful structures in evolving better differential equations to fit a given dataset. Later, they extended this concept to model higher order differential equations for dynamical systems [12]. However, the computational burden is extremely high in all these hybrid methods, which in essence prevents widespread use of this hybridisation philosophy. In order to reduce the computational cost, a new method called Cauchy-guided evolutionary programming (CGEP) method has been developed. To ascertain the potential of the CGEP method, it has been tested on some important benchmark functions. The CGEP method is described in Section 3.

Further improvements in modelling performance are likely to be achieved by a hybrid approach including both GP and CGEP methods. The sole aim of hybridisation is to develop better algorithms for the automatic generation of mathematical models by preserving possibly the best structures in a population pool. As a first step towards the hybridisation of GP and CGEP, the parameters of the GP individuals are fed to the CGEP for further optimisation. However, this increases the execution time tremendously, and it is therefore not suitable for all applications, particularly real time applications. Hence, in the proposed hybrid GP and EP, which is named here as the hybrid genetic evolutionary programming (HGEP) method, the best individual of any GP generation is further optimised by CGEP to better exploit the underlying structure of the best individual tree. However, in order to further minimise the execution time, the CGEP is used only for a few iterations. The HGEP method is described in detail in Section 4.

Section 5 establishes the efficacy of the proposed hybrid evolutionary algorithm for modelling standard symbolic regression problems. In Section 6, a standard model of a simple robotic system is described. Then, the simple robotic manipulator test problems and the details of the experimental set-up are described. The potential results of the experiments on modelling robotic manipulators are illustrated in Section 7. Section 8 discusses the results thoroughly. Finally, in Section 9, conclusions of this paper are presented.

## 2. Evolutionary computation methods

In general, evolutionary computation methods are a very rich class of multi-agent stochastic search (MASS) algorithms based on the neo-Darwinian paradigm of natural evolution, which can perform exhaustive searches in complex solution space. These techniques start with searching a population of feasible solutions generated stochastically. Then, stochastic variations are incorporated into the parameters of the population in order to evolve the solution to a global optimum. Thus, these methods provide a rigorous search in the entire search domain, taking into account the maximum possible interactions among them. The field of research in these evolutionary methods broadly covers three distinct areas: genetic algorithms [13], evolution strategies (ES) [14], and EP [15,16]. The widely used genetic algorithms model evolution based on observed genetic mechanisms, i.e. gene level modelling. Whereas, evolution strategy algorithms model evolution of individuals to better exploit their environment, and use a purely deterministic method of selection. EP algorithms model evolutions of individuals of multiple species competing for shared resources, and essentially utilise stochastic selection.

### *2.1. Evolutionary programming*

An evolutionary programming method, which models evolution at the level of competing species for the same resources, uses mutation as the sole operator for the advancement of generation, and the amount of exploitation and exploration is decided only through the mutation operator. Usually, in the basic EP (BEP) [15], the mutation operator produces one offspring from each parent by adding a Gaussian random variable with zero mean and a variance proportional to the individual fitness score. The value of standard deviation, which is the square root of the variance, decides the characteristics of offspring produced with respect to its parent. A standard deviation close to zero will produce offspring that have more probability of resembling their parent, and much less probability of being largely or altogether different from it. As the value of the standard deviation departs from zero, the probability of resemblance of offspring with their parent decreases, and probability of producing altogether different offspring increases. With this feature, the standard deviation essentially maintains the trade-off between exploration and exploitation in a population during the search.

In recent years, there has been much effort to increase the overall performance of EP in a variety of problem domains. Historically, the normal (Gaussian) distribution is used for the generation of mutation vectors. However, very recently, Cauchy distribution is proposed as a viable alternative to normal distribution. Yao et al. [17] have shown that Cauchy mutation provides faster and better results in comparison to Gaussian mutation for multi-modal functions with many local minima. This enhanced performance with Cauchy distributions is believed to be due to their much longer and flatter tails, which provide a longer step size during the search operation.

### *2.2. Fast EP*

The well-established self-adaptive EP methods work by evolving simultaneously all the object variables and their corresponding mutation parameters or step size. A particular set of mutation parameters associated with an individual survives only when it produces better object variables. The most common variant of self-adaptive EP is the canonical self-adaptive EP (CEP), which is modified with the use of Cauchy distribution ($C(0, 1)$) and is known as fast EP (FEP) [17,18], which updates the $n$ dimensional mutated parameter vector $\boldsymbol{\eta}_i$ and the corresponding object variable vector $\boldsymbol{p}_i$ as per the following equations:

$$\eta_{ij}(k+1) = \eta_{ij}(k)\exp(\tau N(0,1) + \tau' N_j(0,1))$$

$$p_{ij(k+1)} = p_{ij}(k) + \eta_{ij}(k+1)C_j(0,1)$$

where $\eta_{ij}$ and $p_{ij}$ are the $j$th component of the $i$th mutated vector and $j$th object variable of the $i$th individual, respectively, and the exogenous parameters $\tau$ and $\tau'$ are set to $\left(\sqrt{2n}\right)^{-1}$ and $\left(\sqrt{2\sqrt{n}}\right)^{-1}$, respectively [19].

### 2.3. Genetic programming

Genetic programming is a stochastic adaptive search technique in the field of automatic programming (AP) that evolves computer programs to solve or approximately solve problems [20,21]. The evolving individuals are themselves computer programs that are often represented by tree structures (other representations also exist) [21,22]. In the tree representation, the individual programs are represented as rooted trees with ordered branches. Each tree is composed of functions as internal nodes and terminals as levels of the problem. Syntactically correct programs are generated by the use of any programming language like LISP, C, and C++ that represent programs internally as parse trees. Koza [20] used LISP, which has the property that the functions can easily be visualised as trees with syntax in prefix form, and the syntax is preserved by restricting the language to suit the problem with an appropriate number of constants, variables, statements, functions, and operators. A particular problem is solved by this restrictive language formed from a user-defined *terminal set* $\boldsymbol{T}$, that may consist of system inputs, ephemeral constants, and other constituents to solve a task at hand, and *function set* $\boldsymbol{F}$, that usually consists of arithmetic operators in any problem-specific functions. Each function in the function set must be able to accept gracefully, as arguments, the return value of any other function and any data type in the terminal set. The functions and terminals are selected a priori in such a way that they will provide a solution for the problem at hand.

GP starts with an initial population of randomly generated tree-structured computer programs, as discussed above. A fitness score is assigned to each individual program, which evaluates the performance of the individual on a suitable set of test cases. Further, each individual undergoes variations to procreate new evolved individuals by using any of the above-mentioned evolutionary computing methods such as GA, ES and EP. Then, a selection criterion is fixed to select individuals for the next generation [19].

## 3. Cauchy-guided EP

### 3.1. Concepts

Basic evolutionary programming generates one offspring from each parent in a population pool by adding a Gaussian random variable of mean zero and variance proportional to the fitness score of the parent. Then, a stochastic competition selects effective parents for the next generation. In contrast, the CGEP replaces the normally distributed variations of BEP with a Cauchy distributed variation. However, a Cauchy distribution does not posses a finite expected value or standard deviation for order less than 1. Hence, a standard form of Cauchy distribution denoted by $C(0,1)$ and its probability density function (PDF) and cumulative distribution function (CDF) are represented as follows [23]:

$$\text{PDF} = \pi^{-1}(1+x^2)^{-1}c \tag{1}$$

$$\text{CDF} = \tfrac{1}{2} + \pi^{-1}\tan^{-1} x \tag{2}$$

In essence, $C(0,1)$ by itself does not have the capability to carry any problem-specific knowledge of the task at hand. Therefore, $C(0,1)$ alone cannot add any knowledge to the solution process, but its random distribution can be utilised fruitfully to guide the solution in the apparently right direction towards the global optimum. So, one problem dependent deterministic factor $\gamma$ has been formulated, which is then used along with the randomness of $C(0,1)$ to escape from the local optima. Hence, there is increased probability of directing the solution process towards the global optimum. Here, $\gamma$ is selected such that it is directly proportional to the square root of the fitness score and inversely proportional to the problem dimensions, and is defined for the $i$th individual $p_i$ in a population of $m$ individuals

$$\gamma_i \propto \frac{1}{n}\sqrt{f(p_i)} = \frac{\alpha}{n}\sqrt{f(p_i)} \tag{3}$$

where $f(p_i)$ is the fitness score associated with the $i$th individual and $0 < \alpha < 1$ is a proportionality constant, and $n$ is the problem dimension. Hence, the search step size $\Delta x_{ij}$ can be represented as follows:

$$\Delta x_{ij} = C_j(0,1)\frac{\alpha}{n}\sqrt{f(p_i)} \tag{4}$$

Then, the $i$th offspring generated from the $i$th individual $p_i$ can be represented as follows:

$$p_{ij} + C_j(0,1)\frac{\alpha}{n}\sqrt{f(p_i)} \tag{5}$$

where $C_j(0,1)$, $j = 1, 2, \ldots, n$, represents the Cauchy random variate for the $j$th variable of the $i$th individual.

### 3.2. Performance of CGEP

To test the performance of the CGEP over that of canonical EP (CEP) and fast EP, a set of eight most typical function minimisation problems from the benchmark functions [17,20] has been considered. These functions are:

$$f_1(x) = \sum_{i=1}^{n} x_i^2, \quad -100 \le x_i \le 100$$

$$f_2(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1),$$
$$-32 \le x_i \le 32$$

$$f_3(x) = \sum_{i=1}^{n-1}\{100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\},$$
$$-30 \le x_i \le 30$$

$$f_4(x) = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i|, \quad -10 \le x_i \le 10$$

$$f_5(x) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$$

where

$$[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & \cdots & 16 & 32 \\ -32 & -32 & -32 & -32 & \cdots & 32 & 32 \end{bmatrix},$$

and $-65.536 \le x_i \le 65.536$

$$f_6 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{\mathrm{i}}}\right) + 1,$$
$$-600 \le x_i \le 600$$

$$f_7 = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2, \quad -100 \le x_i \le 100$$

$$f_8 = \sum_{i=1}^{n}\{x_i^2 - 10\cos(2\pi x_i) + 10\},$$
$$-5.12 \le x_i \le 5.12$$

Function $f_1$ is a generalised unimodal sphere function with a minimum at $X = (0, \ldots, 0)$. Function $f_2$ is Ackley's continuous unimodal test function, obtained by modulating an exponential function with a cosine wave of moderate amplitude. The term $20 + \exp(1)$ is added to move the global optimum function value to zero at $X = (0, \ldots, 0)$. Function $f_3$, the generalised Rosenbrock's saddle, has a very steep valley along $x_{i+1} = x_i^2$ with the global minimum function value of zero at point $X = (1, \ldots, 1)$. Both the Schwefel's function $f_4$ and function $f_7$ have a unique optimal function value of zero at $X = (0, \ldots, 0)$. Function $f_5$ is typical of De Jong's five-function test-bed and contains multiple local optima. This is known as Shekel's Foxholes, and is stated to be very pathologic. It has the global optimum function value of 0.998004. Griewangk's function $f_6$ has many local minima, which usually misleads the solution from the global minimum $f_6(0, \ldots, 0) = 0.0$. Function $f_8$ is a generalised Rastrigin's function with the optimum function value of zero at the origin. All of these functions except $f_5$ have 30 dimensions, whereas $f_5$ has two. The typical behaviours of these functions are described in Yao et al. [17].

All the experiments were performed with a population size of 100 and a tournament size 10. All experiments on these functions were averaged over 50 independent trials. In the event of the fitness score

Table 1
Comparisons of average best score between CEP, FEP and CGEP

| Function | Generation | $\alpha$ | CEP | FEP | CGEP |
|---|---|---|---|---|---|
| $f_1$ | 1500 | $5.5 \times 10^{-3}$ | $5.75 \times 10^{-2}$ | $1.10 \times 10^{-4}$ | $2.43 \times 10^{-16}$ |
| $f_2$ | 2000 | $8.0 \times 10^{-5}$ | $2.58 \times 10^{-3}$ | $2.06 \times 10^{-1}$ | $3.72 \times 10^{-5}$ |
| $f_3$ | 5000 | $1.0 \times 10^{-2}$ | 4.227 | 1.042 | $2.20 \times 10^{-6}$ |
| $f_5$ | 20000 | $3.0 \times 10^{-5}$ | $1.03 \times 10^{01}$ | 5.444 | 4.00 |
| $f_9$ | 5000 | $1.1 \times 10^{-3}$ | $1.07 \times 10^{02}$ | $1.227 \times 10^{2}$ | 0.0 |
| $f_{10}$ | 1500 | $1.5 \times 10^{-3}$ | 9.675 | $4.79 \times 10^{-3}$ | $9.88 \times 10^{-4}$ |
| $f_{11}$ | 2000 | $1.0 \times 10^{-2}$ | $8.35 \times 10^{-1}$ | $6.29 \times 10^{-2}$ | $9.76 \times 10^{-3}$ |
| $f_{14}$ | 100 | $1.0 \times 10^{-2}$ | 1.857361 | 1.407797 | 0.998004 |

in the CGEP method being negative, it was set to an arbitrarily low value of 0.01. FEP and CEP were implemented with a fixed lower bound of 0.0001 on the mutation parameters. In all the cases, CGEP either outperformed or at least yielded comparable results with those of the FEP and CEP. This is illustrated in Table 1. This shows the potential of CGEP, which is efficient yet computationally and functionally simple. The parameters used in the simulation of the CGEP algorithm are also shown in Table 1. Only one parameter $\alpha$ needs to be fine-tuned within a very small tuning domain for all the eight test functions. From the experience with the tuning of this algorithm, it may be inferred that it is good practice to start the tuning with a search limit of $10^{-5}$ and progressively check at multiples of 10. It has been observed that the optimal value of $\alpha$ lies near these steps. This clearly verifies the robustness of the proposed algorithm. It is noteworthy to observe the test results of function $f_5$, Shekel's Foxholes, which contains a moderate number of local optima. Usually, all the variants of EP reported in the literature exhibit constantly degraded performance to arrive at the global minimum and often stall at local minima. But, interestingly enough, CGEP finds the exact global optimum value of 0.998004, thereby proving its superb capacity for optimisation tasks. On all other functions, the CGEP's performance is also excellent. These performances remain consistent with many other functions.

In essence, the CGEP is a simple variant of BEP. The major difference between CGEP, CEP and FEP is that the former is a non-self-adaptive method, whereas the latter two are self-adaptive. All the three methods, CGEP, FEP and CEP use Cauchy distribution to evolve the object variables. The CGEP's learning ability depends on the selection of a fixed initial learning coefficient. Now, once the efficacy of the CGEP method has been established, with the added advantage of its simplicity over CEP and FEP, it is ready to be used for the parameter optimisation of the parameters of the best tree in every generation for the GP.

## 4. Description of HGEP algorithm

The hybrid genetic evolutionary programming is based on a steady-state GP. In each generation, two potent tree individuals from the population pool consisting of all the tree individuals $P_t$ are selected independently by means of tournament selections with a tournament size $t_c$. In a tournament of size $t_c$, all the individuals compete with each other and finally the winner is selected amongst them. Then, the selected tree individuals generate offspring by using the simple subtree crossover mechanism. Subsequently, the offspring replace the two worst parents by using a similar tournament method. Thereafter, a mutation operation is performed depending on the problem complexities. For complex problems, it is very often necessary to perform a mutation operation to generate possibly quite different solution trees. The mutation operator selects worst tree individuals from the independent tournament of size $t_m$. Then, each of the selected individuals are mutated through a series of randomly selected mutation operations $\bar{M}_0$ from the set $\bar{M}$, where $\bar{M} \subseteq$ {OneNode, RandNodes, Swap, Grow, Trunc, Gaussian}. These mutation operators work as follows: (i) the *OneNode* mutation operator replaces a randomly selected tree node with another random node of the same arity; (ii) in *RandNode*s, a particular node in a tree is replaced by a random node of the same arity with a very low probability; (iii) the *Swap* mutation operator swaps the arguments of a randomly selected node of arity more than one; (iv) then, the *Grow* operator selects a terminal node randomly, and replaces that with a randomly generated subtree of pre-specified size; (v) the *Trunc* mutation operator randomly selects a function node and replaces the complete subtree starting at that node with a terminal node; (vi) then, the *Gaussian* operator acts on a randomly selected numerical terminal node, and perturbs that with a Gaussian variate of mean zero and standard deviation 0.1. The number of mutation operators are selected as per a Poisson variate. To gain a better understanding of the action of these mutation operators, let the Poisson variate be 3. Now, three mutation operators will be selected randomly with replacement from the mutation set $\bar{M}$. Suppose, the randomly selected three mutation operators are *OneNode*, *Grow* and *Gaussian*. Then, the action of all the three mutation operators on a pre-selected tree individual $p_{\mathrm{t}}^{\mathrm{m}}$ to procreate an offspring $\mathrm{o}_{\mathrm{t}}^{\mathrm{m}}$ can be given as follows:

$$o_{\mathrm{t}}^{\mathrm{m}} = \mathrm{Gaussian}(\mathrm{Grow}(\mathrm{OneNode}(p_{\mathrm{t}}^{\mathrm{m}}))),$$

where the superscript ‘m’ indicates that it is a mutation related operation. Here, the offspring directly replaces its parent.

Now, the best individual $p_{\mathrm{t}}^{\mathrm{b}}$ in the population pool $P_{\mathrm{t}}$ is selected. Then, CGEP works on a single individual $p_1 = \{p_{1j} | j \in (1, \dots, n_0)\}$, which is formed with the numerical nodes (constant terminal nodes) of $p_{\mathrm{t}}^{\mathrm{b}}$. Thus, the size $n_0$ of this individual is not a constant, but depends on the number of numerical nodes in the best tree individual $p_{\mathrm{t}}^{\mathrm{b}}$. Subsequently, $\lambda$ offspring are generated from $p_1$ by perturbing its object variables by a Cauchy variate of mode zero and median $(\alpha/n_0)\sqrt{f(p_1)}$. Then, a stochastic tournament selection with ‘$c$’ number of competitors is used to decide the individual to be used to construct the new and more potent individual tree $p_{\mathrm{t}}^{\mathrm{n}}$ with the same structure as that of $p_{\mathrm{t}}^{\mathrm{b}}$. Finally, $p_{\mathrm{t}}^{\mathrm{n}}$ replaces the worst tree individual in the entire population pool of tree individuals $P_{\mathrm{t}}$. Then, the new pool of tree individuals after crossover, mutation and CGEP optimisation is ready for the next generation. The complete pseudo-code of the HGEP algorithm is shown in Table 2.

## 5. Hybrid genetic evolutionary programming for symbolic regression

### 5.1. Test problems

After the development of the HGEP algorithm, its performance on standard problems must be tested before it can be applied to any complex non-linear system. For the verification of the potential of HGEP, two test problems of symbolic regression have been chosen. The first test problem is a single input regression problem, whereas the second one is a two-input regression problem. The second problem was selected to study the effects of multiple inputs on the performance of different GP-based algorithm. Both these test problems are described below.

#### 5.1.1. Test problem 1: one-input symbolic regression

Given a finite input–output dataset from the equation, $y = 2.719x^2 + 3.14161x$, the goal is to find the underlying model from the dataset.

#### 5.1.2. Test problem 2: two-input symbolic regression

Given a finite input–output dataset from the equation, $z = x^2 + y^2$, the goal is to find the underlying model from the dataset.

### 5.2. Experimental set-up

#### 5.2.1. Test problem 1: one-input symbolic regression

For this problem, 20 input–output data points in the range between $-1$ and 1 were selected randomly. Generation of random tree structures and subsequent manipulation of these trees was performed using steady-state GP and HGEP method. Initial trees were generated using a grow method. The function and terminal sets were: $\boldsymbol{F} = \{+, -, \times, /\}$ and $\boldsymbol{T} = \{x, U(-1, 1)\}$, where $U(-1, 1)$ is a random number between $-1$ and 1. The parameters used for the GP part of the HGEP for this problem are shown in Table 3 and the parameters for the CGEP to optimize the ephemeral constants are shown in Table 4. CGEP used a stochastic $(1 + 10)$ selection method.

#### 5.2.2. Test problem 2: two-input symbolic regression

Similar to the single-input regression problem, a steady-state GP and HGEP were used. The initial

Table 2
Pseudo-code of hybrid genetic evolutionary programming (HGEP) algorithm

```
Given:
  GP parameters:
  A function set $\boldsymbol{F}$ and a terminal set $\boldsymbol{T}$, probability of crossover, probability of mutation for all node mutation, probability of
    terminal node selection, Poisson mean, tournament size for parent selection for crossover ($t_c$) and mutation ($t_m$), tree
    initialisation method (grow, full or ramped-half-and-half), tree population size $\mu_t$, number of fitness cases, maximum tree
    depth after crossover/mutation, maximum size of a mutant subtree
  EP parameters:
  Learning coefficient $\alpha$, EP population size $\mu$, number of iterations, number of offspring $\lambda$, number of competition $c$
Step 1. Initialisation:
  Initialise a population pool $P_t$ for the GP, consisting of tree individuals
  for $i := 1$ to $\mu_t$ do
    $p_t[i]$:= randomly generate tree individuals;[a]
  where $p_t[i]$ represents the $i$th individual tree;

Step 2. Evaluation:
  Evaluate the individuals
  for $i := 1$ to $\mu_t$ do
    Evaluate ($p_t[i]$); //assign a fitness value to each tree individual

Step 3. Recombination (subtree crossover):
  (i) Select two parents for crossover
    Select randomly a group of $t_c$ individuals comp[$i$] $\forall i \in \{1, \ldots, t_c\}$ from the population pool $P_t$
    for $i := 1$ to $t_c$ do
      ir = an integer random number $\in U(0, \mu_t)$;
        comp[$i$] = $p_t$[ir];
    $p_t^{c1}$ = comp[1];
    for $i := 1$ to $t_c$ do
      if $(f(p_t^{c1}) < f(\mathrm{comp}[i]))\, p_t^{c1} = \mathrm{comp}[i]$;
    Similarly, select the second parent for crossover $p_t^{c2}$;
  (ii) Select crossover sites
$cn1$ = randomly select a node on $p_t^{c1}$
$cn2$ = randomly select a node on $p_t^{c2}$
  (iii) Exchange subtrees with root nodes $cn1$ and $cn2$ between $p_t^{c1}$ and $p_t^{c2}$ to generate two offspring $o_t^{c1}$ and $o_t^{c2}$
  (iv) Replacement of two tree individuals from $P_t$ with offspring $o_t^{c1}$ and $o_t^{c2}$
    Select the worst individual to be replaced
    ir1 = an integer random number $\in U(0, \mu_t)$;
    for $i := 1$ to $t_c$ do
      do
        ir2 = an integer random number $\in U(0, \mu_t)$;
      while(ir1 $\neq$ ir2);
      if$(f(p_t[\mathrm{irl}]) > f(p_t[\mathrm{ir2}]))$ir1 = ir2;
      $p_t[\mathrm{ir1}] = o_t^{c1}$; //replace one individual
    Repeat the above process to place $o_t^{c2}$ in the pool $P_t$

Step 4. Mutation:
  (i) Select parents for mutation operation[b]
    Select randomly a group of $t_m$ individuals comp[$i$] $\forall i \in \{1, t_m\}$ from the population pool $P_t$
      for $i := 1$ to $t_m$ do
        ir = an integer random number $\in U(0, \mu_t)$;
        comp[$i$] = $p_t$[ir];
        $p_t^{m1}$ = comp[1];
        for $i := 1$ to $t_m$ do
          if$(f(p_t^{m1}) < f(\mathrm{comp}[i]))\, p_t^{m1} = \mathrm{comp}[i]$;
        Similarly, select the second parent for mutation $p_t^{m2}$
  (ii) Calculate the number of mutation operations $\gamma_p$ from a Poisson distribution with a given mean
  (iii) Generate offspring by mutation
```

Table 2 (*Continued*)

```
    while($\gamma_p \neq 0$)
      Randomly select a mutation operator $M_0$ from the mutation operator set
        $\bar{M}|\bar{M}_0 \in \bar{M} \subseteq$ {OneNode, RandNodes, Swap, Grow, Trunc, Gaussian};
      $o_t^{m1} = \bar{M}_0(p_t^{m1})$;
      $\gamma_p = \gamma_p - 1$;
      Similarly, generate the second offspring $o_t^{m2}$ by mutation;
  (iv) Replacement of offspring
    Each offspring replace its parent
      $p_t^{m1} = o_t^{m1}$;
      $p_t^{m2} = o_t^{m2}$;

Step 5. *EP calculations*:
  (i) Select the best tree individual
    $p_t^b = p_t[1]$;
    $i := 2$ to $\mu_t$ do
      if$(f(p_t^b) < f(p_t[i]))\, p_t^b = p_t[i]$;
  (ii) Gather the parameters of $p_t^b$ as a row vector p[1]:{p[1]|p[1][j], $\forall j \in \{1, \dots, n_o\}$} with $n_o$ the number of parameters in the tree $p_t^b$
  (iii) Evaluate ($p$[1]); //assign a fitness value
  (iv) Generate λ offspring from $p$[1] by mutation operation as described below
  *Mutation*:
  *Calculate*:
  for $j := 1$ to $n_o$ do
    $s[j] := \frac{\sqrt{f(p_t^b)}}{n_o}$;;
    where $s[j]$ is the scale factor for the $j$th element, and $p_t^b$ the fitness of the best tree individual $p_t^b$, which maps $p_t^b \to 3$
    *Mutate*:
    for $i := 1$ to λ do
      for $j := 1$ to $n_o$ do
        $C$=generate a Gaussian random number $C(0, 1)$;
        $p[i + 1][j] = p[1][j] + Cs[j]$;
    where $C(0, 1)$ represents a standard Cauchy variable
    *Evaluation*:
    Evaluate the offspring
    for $i := 1$ to λ do
    for $j := 1$ to $n_o$ do
    *Evaluate*($p[i][j]$); //assign fitness equal to the fitness with parameters of $p_t$ and the structure of $p_t^b$
    *Selection*:
    Select one individual from $(\lambda + 1)$ individuals by using a stochastic competition comprising of '$c$' number of participant individuals
      for $i := 1$ to $(\lambda + 1)$ do
      wt[$i$] := 0.0;
      for $j := 1$ to $c$ do
        $t := [(\lambda + 1)U(0, 1)]$;
      if $f(p[i]) \leq f(p[t])$
        then wt[$i$] := $wt[i] + 1$;
      where $[\cdot]$ denotes a greatest integer function
      for $i := 1$ to $(\lambda + 1)$ do
        for $j := 1$ to $n_o$ do
          *Select* the best individual according to its weight wt (i.e. number of wins)
Step 6. If termination condition is not reached return to step 3
```

[a] Here, each individual consists of one tree. However, for multiple output systems, parallel trees can be generated analogous to the number of object variables per individual in a real coded evolutionary algorithm.

[b] The number of parents to be mutated can be any number between 0 and $\mu_t$. However, for this research work, two parents are considered to undergo mutation.

Table 3
GP parameter values for one-input regression analysis

| Parameters | Values |
|---|---|
| Population size | 300 |
| Probability of leaf selection | 0.5 |
| Maximum tree depth after crossover | 17 |
| Probability of mutation | 0 |
| Number of crossover operations per generation | 2 |
| Tournament size | 4 |
| Number of fitness cases | 20 |
| Initial tree depths | 5 |
| Crossover probability | 1.0 |
| Number of generations | 200 |

Table 4
CGEP parameter settings for regression analysis

| Parameter | Value |
|---|---|
| Population size | 1 |
| Tournament size | 4 |
| Learning coefficient ($\alpha$) | 0.0005 |
| Number of offspring | 5 |
| Number of iterations | 3 |

Table 5
GP parameter values for two-input regression analysis

| Parameter | Value |
|---|---|
| Population size | 300 |
| Probability of leaf selection | 0.5 |
| Maximum tree depth after crossover | 17 |
| Probability of mutation | 0 |
| Number of crossover operations per generation | 2 |
| Tournament size | 4 |
| Number of fitness cases | 20 |
| Initial tree depths | 5 |
| Crossover probability | 1.0 |
| Number of generations | 400 |

training set of 20 independent data points was selected randomly within the open interval $(-1, 1)$. The parameters used for the GP part of the HGEP are illustrated in Table 5. However, the parameters used for the EP part of HGEP are kept the same as for single-input regression, as shown in Table 4.

The fitness of an individual tree has been calculated as the sum-square error of all the fitness cases used to build the model. The overall fitness can be expressed as follows:

$$\mathrm{Fit}_{\mathrm{nor}} = \frac{1}{1 + \mathrm{Fit}_{\mathrm{ind}}}$$

where $\mathrm{Fit}_{\mathrm{nor}}$ is the normalised overall fitness and individual tree fitness

$$\mathrm{Fit}_{\mathrm{ind}} = \sum_{i=1}^{m} e_i^2$$

with $m$ being the number of fitness cases.

## 6. Results

### 6.1. Test problem 1: one-input symbolic regression

All the results of HGEP have been compared with a simple GP method. The average best and average mean results of 10 independent runs of HGEP and GP are shown in Table 6. The corresponding $t$-test results are also included in Table 6. The $t$-test results indicate the statistical significance of the results obtained from the GP and HGEP methods in 10 different trials. When the $t$-values are negative it suggests that the first of the two methods under test is better than the second one. The convergence characteristics of the average best and average mean results of both HGEP and GP are shown in Fig. 1. It is clear from Fig. 1 that

Table 6
Average best and average mean results of HGEP and GP of one-input regression problem

| HGEP | | GP | | $t$-test (GP–HGEP) | |
|---|---|---|---|---|---|
| Average best | Average mean | Average best | Average mean | Average best | Average mean |
| 0.984433 (0.0474164) | 0.822811 (0.0345126) | 0.868407 (0.124908) | 0.323265 (0.0338958) | 3.68467 (0.00503883) | 31.4079 (1.64962E−10) |

The bracketed quantities indicate the standard deviation for HGEP and GP, and for $t$-test, these indicate significance values.

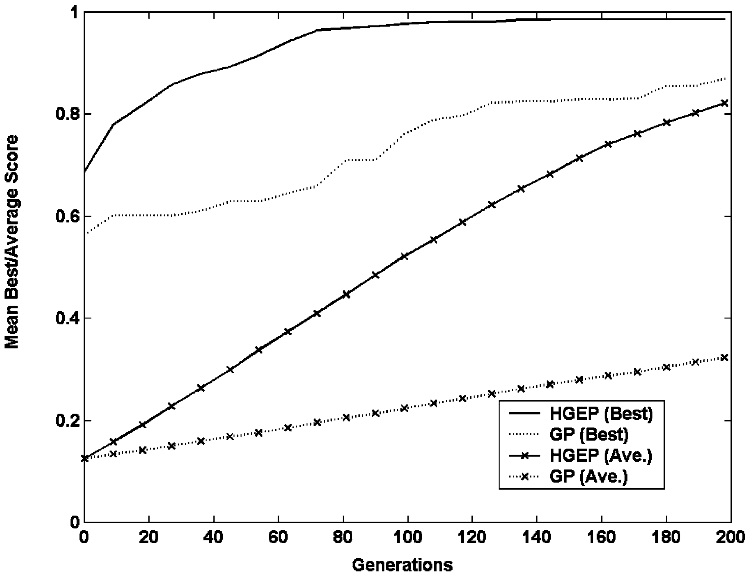

Fig. 1. Average best and mean characteristics of HGEP and GP methods for one input symbolic regression problem.

both average best and average mean results of HGEP are exceedingly better than simple GP. Out of 10 independent runs, HGEP yield the best fitness score corresponding to the actual model, which is one, in eight cases. In contrast, simple GP did not succeed in any of the runs to achieve this value. It is interesting to note that, in all the eight cases out of the 10 runs, although the best tree individuals have the same fitness value of one, their structures are completely different from each other. This in turn implies that, for a particular dataset, there could be many valid models. Now, three of the prominent results of HGEP are depicted below for further examination.

Trial 6: The program tree in prefix notation can be represented as follows:

$$(/(/(x(/(/(0.367910,\ -(x-1.155566))\\ -(x-0.000041))x)$$

and this can be simplified to

$$y=\frac{(x+1.155566)(x+0.000041)}{0.367910}\\ =2.718056x^2+3.14078171x\\ +1.2877662\times 10^{-0.4}$$

Trial 7: Here, the program tree in prefix notation can be represented as follows:

$$(+(-(x(-(^*(x-1.141644))\\ \times(^*(/(-(x(^*(x-0.311497)))0.482349)x))x)$$

This can be simplified into

$$y=2x+1.141644x+\frac{1.311497x^2}{0.482349}\\ =2.7189794x^2+3.141644x$$

Trial 8: Here, the program tree in prefix notation can be represented as follows:

$$(^*(/(+(^*(-0.352884x)-0.407616)-0.259527)\\ \times(+(xx))))$$

This can be simplified into

$$y=\left(\frac{-0.352884x-0.407616}{-0.259527}\right)2x\\ =2.7194396x^2+3.1412223x$$

In all the three cases discussed above, the resultant models represent the exact model very closely. Out of these, the results in trial 7 are the best in terms of its accuracy, but the number of nodes is the largest. This suggests that minimisation of the number of nodes to represent a problem may not be a good option in the broader field of tree-based program optimisation.

On the other hand, the best tree in all the 10 independent runs with simple GP was produced in the sixth trial with best fitness value of 0.968983. The corresponding tree individual has been presented below.

The program tree in prefix notation can be represented as follows:

$$(+(-(+(xx)^{*}(-(0.106337x)(+(-0.381775 \times(/(x\,0.402704))))))(-(/(x\,0.472469)0.060648))))$$

This can be simplified into

$$y = 2.4832135x^2 + 3.0151025x - 0.0200511$$

Thus, it is clear from these results that HGEP outperforms simple GP on a one-input symbolic regression problem.

### 6.2. Test problem 2: two-input symbolic regression

For the two-input symbolic regression problem, the number of generations was increased to 400 as compared to 200 in a one-input symbolic regression problem. This was essential because problem complexity increases with the increase of the number of inputs. Increasing the number of inputs expands the possible search space exponentially. Thus, two-input regression problems are seemingly more complex than the single-input regression problems. The average best and average mean results averaged over 10 independent runs for both HGEP and GP are presented in Table 7. All the statistical results are also included in Table 7. Here, it is very important to observe that the average best result at the end of the 400th generation in the case of GP is better than HGEP. The *t*-test results indicate that the average best results of HGEP and GP are not significantly different, whereas the average mean results of HGEP are much better than those of GP. However, the convergence characteristics of both HGEP and GP, as shown in Fig. 2, indicate that the average best results of HGEP are consistently slightly better than GP over most of the generations except towards the end. Table 8 examines the fitness values of the best individual at the 400th generation for all the 10 runs. It can be seen that the best fitness values in all the runs of HGEP are widely scattered compared to those of GP. Also, it is worth noticing that the best fitness value corresponding to the exact model, i.e. one, is only obtained once in HGEP, whereas in GP it is not reached in any of the 10 runs. This shows that the HGEP found a closely matched model but GP could not. Now, the best tree in all the 10 runs of HGEP and GP are described next.

The best tree of HGEP at trial 1 with a fitness value of one has been given as follows:

$$(+(^{*}(yy))(^{*}(xx)))$$

This can be simplified to

$$z = x^2 + y^2$$

where $z$ is the output. Thus, it exactly represents the actual model.

Table 8
Average best results of HGEP and GP in all the 10 trials

| Trial | HGEP | GP |
|---|---|---|
| 1 | 1.000000 | 0.952590 |
| 2 | 0.977641 | 0.964483 |
| 3 | 0.966503 | 0.962350 |
| 4 | 0.934917 | 0.969220 |
| 5 | 0.934917 | 0.942918 |
| 6 | 0.896411 | 0.951134 |
| 7 | 0.946584 | 0.964510 |
| 8 | 0.985854 | 0.963643 |
| 9 | 0.983452 | 0.946353 |
| 10 | 0.946584 | 0.973827 |

Table 7
Average best and average mean results of HGEP and GP of two-input regression problem

| HGEP | | GP | | *t*-test (GP–HGEP) | |
|---|---|---|---|---|---|
| Average best | Average mean | Average best | Average mean | Average best | Average mean |
| 0.957286 (0.0311985) | 0.940225 (0.266465) | 0.959103 (0.010246) | 0.875574 (0.0337756) | −0.176124 (0.864096) | 4.14208 (0.00251386) |

The bracketed quantities indicate the standard deviation for HGEP and GP, and for *t*-test, these indicate significance values.

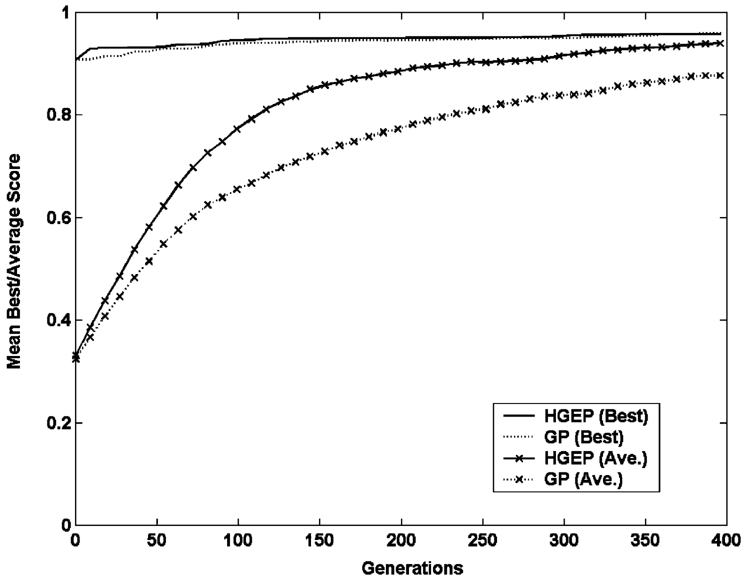

Fig. 2. Average best and mean characteristics of HGEP and GP methods for two-input symbolic regression problem.

The best tree of GP at trial 10 with fitness score 0.973827 has been shown below:

$$\begin{aligned}&({}^*(-(+(y(-(xx)))({}^*({}^*(x({}^*(-0.239596y)))\\&\quad\times({}^*(0.126988x))))) + ({}^*x({}^*(x(+({}^*(x^*(xx)))\\&\quad+(y({}^*({}^*(0.022461(-(xx))))(+({}^*(-0.239786x)y))\\&\quad\times(+(y({}^*({}^*(0.022461y)(+({}^*(-0.239786x))y))))\end{aligned}$$

This can be simplified to

$$z = x^5 + yx^2 + y - 0.053858333xy + 0.22461y^2$$

Although the fitness is not far from the actual model fitness of one, the resulting model is hopelessly bad. This shows that the fitness score may not be taken as a confirmation of model accuracy. Also, it is important to note the experimentation also established that even increasing the number of generations by 10 times did not improve the performance of GP.

## 7. Experimental studies on robot modelling

The dynamic model of a $n$-link terrestrial manipulator system with all the masses assumed to be point masses at the distal end of each link and gravity terms, can be expressed as follows [24]:

$$\boldsymbol{T} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) \qquad (6)$$

where $\boldsymbol{T} \in 3^n$ is the driving torque vector, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}} \in 3^n$ are the joint position, velocity and acceleration vectors, respectively, $\boldsymbol{M}(\boldsymbol{q}) \in 3^{n\times n}$ is the mass matrix of the manipulator, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in 3^n$ the centrifugal and coriolis terms and $\boldsymbol{G}(\boldsymbol{q}) \in 3^n$ the gravity vector. Here, the gravity terms are considered just to represent the complete terrestrial system. For a two-link manipulator having link lengths $l_1$, $l_2$ and corresponding mass $m_1$, $m_2$, the matrices and vectors in Eq. (6) can be represented as follows:

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} l_2^2m_2 + 2l_1l_2m_2c_2 + l_1^2(m_1+m_2) & l_2^2m_2 + l_1l_2m_2c_2 \\ l_2^2m_2 + l_1l_2m_2c_2 & l_2^2m_2 \end{bmatrix} \qquad (7)$$

where $c_2 = \cos(q_2)$ and $\boldsymbol{M}(\boldsymbol{q})$ is a symmetric, positive definite matrix.

$$\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} m_2 l_1 l_2 s_2 \dot{q}_2^2 - m_2 l_1 l_2 s_2 \dot{q}_1 \dot{q}_2 \\ m_2 l_1 l_2 s_2 \dot{q}_1^2 \end{bmatrix} \tag{8}$$

where $s_2 = \sin(q_2)$.

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ m_2 l_2 g c_{12} \end{bmatrix} \tag{9}$$

where $c_1 = \cos(q_1)$ and $c_{12} = \cos(q_1 + q_2)$.

### 7.1. *Test problems*

The following two test problems were used to generate the initial dataset for verifying the potential of the proposed HGEP method for extracting the underlying model.

#### 7.1.1. *Test problem 1: single-link robot manipulator*

From Eq. (6), the equations of motion of a single-link manipulator can be expressed as follows:

$$\tau_1 = l_1^2 m_1 \ddot{q}_1 + m_1 l_1 g c_1 \tag{10}$$

Now, the joint acceleration $\ddot{q}_1$ can be represented as follows:

$$\ddot{q}_1 = \frac{1}{l_1^2 m_1}(\tau_1 - m_1 l_1 g c_1) \tag{11}$$

Eqs. (10) and (11) involve less complexities compared to the two-link manipulator described next.

#### 7.1.2. *Test problem 2: forward dynamics model of two-link robot manipulator*

The general form of the equation of motion of a manipulator system is given in Eq. (6). For a two-link manipulator with Eqs. (7)–(9), the forward dynamics equation can be represented as follows:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{T} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})) \tag{12}$$

This is a highly complex modelling problem due to the presence of the non-linear coupling between the parameters of the two links of the manipulator.

### 7.2. *Experimental set-up*

The experiments on the above two problems were performed under quite different conditions. These are discussed below.

Table 9
GP parameter values for single-link manipulator

| Parameter | Value |
|---|---|
| Population size | 100 |
| Probability of leaf selection | 0.5 |
| Maximum depth after crossover | 17 |
| Probability of mutation | 0 |
| Number of crossover operations per generation | 2 |
| Tournament size | 4 |
| Number of fitness cases | 20 |
| Initial tree depths | 5 |
| Number of generations | 300 |
| Number of crossover operations per generation | 0 |

#### 7.2.1. *Test problem 1: single-link robot manipulator*

The dataset consisting of 20 input–output samples was generated from Eq. (11) for a random torque $\tau_1$ between $-5$ and 5. Here, a normal grow initialisation method was used for generating initial tree individuals. The parameters used for the GP part of the HGEP are illustrated in Table 9. Here, the GP part of the HGEP uses a steady-state GP for extracting the model from the supplied data points. The parameters of the CGEP were kept the same as those in Table 4. The experiments were performed on two cases of function sets $\boldsymbol{F}$ and terminal sets $\boldsymbol{T}$. The function and terminal sets are described below.

**Case 1.** $\boldsymbol{F} = \{+, -, \times, /\}$ and $\boldsymbol{T} = \{\tau_1, \cos(q_1), U(-1, 1)\}$, where $U(-1, 1)$ is a random number between –1 and 1.

**Case 2.** $\boldsymbol{F} = \{+, -, \times, /, \cos\}$ and $\boldsymbol{T} = \{\tau_1, q_1, U(-1, 1)\}$.

The parameters of the robotic manipulator system are shown in Table 10.

#### 7.2.2. *Test problem 2: forward dynamics model of two-link robot manipulator*

The model represented in Eq. (12) is a highly complex and coupled system. The values of joint

Table 10
Manipulator parameter setting

| Parameter | Value |
|---|---|
| Mass of each links | 1 kg |
| Acceleration due to gravity | 9.8 m/s$^2$ |
| Lengths of each link | 1 m |

acceleration associated with any link depend on the torque, joint velocity, joint angle and masses and link lengths of both the links of the manipulator. This complicates the selection of the combinations of inputs to form the initial training dataset. It has already been shown that the increase in the number of inputs increases the complexity of the problem and it becomes hard to get the exact model. Because of this, the number of inputs are relatively high in a two-link manipulator problem. In addition, there are equally large numbers of possible input patterns that generate different actions of the manipulator. However, for a manipulator system to perform a particular task, a fixed pattern of joint and end-effector paths must be followed. Moreover, a particular task excites only certain specific modes of the manipulator. Hence, it is very likely that the dataset used for the modelling task is biased towards the specific job for which the data has been collected. This in fact says that it is very unlikely that a general model for a robotic manipulator system can be found. In this work three different input–output datasets are generated and tested for model building.

**Case 1** (A single randomly generated sinusoid). Here joint angles are assumed to follow sinusoidal variations:

$$q^{\mathrm{d}} = q^{\max}(1 - \cos \omega t) \qquad (13)$$

where $q^{\mathrm{d}}$ is the desired joint angle, $q^{\max} \in U(0, 0.2)$ is the maximum value of the sinusoidal joint variation and $\omega \in U(0, 2\pi)$ is the angular frequency of the sinusoid. Here, $q^{\max}$ and $\omega$ were chosen randomly in the prescribed open interval. A total of 40 data points have been generated within a time period of [0,1) for each of the possible inputs for the model building.

**Case 2** (Two randomly generated sinusoids). In this case, 20 data points with each selection of $q^{\max}$ and $\omega$ were generated. Thus, the entire training dataset generated from Eq. (13) total of 40 data points for each input. The choice of this number of inputs was made in order to keep the number of fitness cases fixed at a value of 40 in both the Cases 1 and 2.

**Case 3** (Multiple sinusoids). Here, 20 sinusoids of random amplitude and frequency were selected for each joint angle variation. The sinusoidal equation used here is given as follows:

$$q^{\mathrm{d}} = \tfrac{1}{2}(q^{\max})\cos \omega t \qquad (14)$$

Using Eq. (14), 40 data points (torque) were generated with $q^{\max}$ varying uniformly and randomly between 0 and 3, and $\omega$ varying between 0 and $2\pi$. Thus, a total of 800 input–output data points were used to train the GP trees. Each tree is allowed to be exposed to a particular sinusoid for only two generations. Hence, 80 generations are required for the GP trees to face all the data points in the input–output dataset.

In every generation, two individuals were selected each for crossover and mutation operation. The ephemeral constants of the best tree in each generation are evolved using CGEP. The function and terminal sets for this problem are: $F = \{+, -, \times, /\}$, $T = \{\tau_1, \tau_2, q_1, q_2, \dot{q}_1, \dot{q}_2, \cos(q_1), \cos(q_2), \sin(q_1), \sin(q_2)\}$. Then, initial GP trees are generated using a ramped half-and-half method. The GP parameters are shown in Table 11. The CGEP parameters are set to exactly the same values as those used in regression analysis, and are tabulated in Table 4.

Table 11
Parameter values for the GP part of the HGEP for two-link manipulator

| Parameter | Value |
|---|---|
| Population size | 300 |
| Probability of leaf selection | 0.5 |
| Maximum depth after crossover | 17 |
| Probability of mutation in RandNode | 0.05 |
| Number of crossover operations per generation | 2 |
| Tournament size | 4 |
| Number of fitness cases | 40 |
| Initial tree depths | (8, 7, 6, 5, 4) |
| Number of generations | 16000 |
| Number of mutation operations per generation | 2 |

## 8. Results

It has already been discussed that HGEP outperforms simple GP on both symbolic regression test problems. Hence in this section, both HGEP and GP are considered for the one-link manipulator, whereas

Table 12
Average best and average mean results of HGEP and GP for one-link robotic manipulator system

| Case studies | HGEP | | GP | | *t*-test (GP–HGEP) | |
|---|---|---|---|---|---|---|
| | Average best | Average mean | Average best | Average mean | Average best | Average mean |
| Case 1 | 0.923908 | 0.782405 | 0.639269 | 0.073540 | 2.01015 | 9.24509 |
| | (0.240179) | (0.234272) | (0.297452) | (0.036198) | (0.0753096) | (6.85E−06) |
| Case 2 | 0.470207 | 0.344297 | 0.207283 | 0.036934 | 1.98864 | 3.1308 |
| | (0.335021) | (0.318587) | (0.275412) | (0.016351) | (0.0779675) | (0.0120616). |

The bracketed quantities indicate the standard deviation for HGEP and GP, and for *t*-test, these indicate significance values. Case 1 is with pre-processed inputs and Case 2 is without any pre-processed inputs.

only HGEP has been used to extract the model for the complex two-link manipulator system. For all the tests, the best result of the 10 independent runs of HGEP and GP has been chosen as the final model of the system.

### 8.1. Test problem 1: single-link robot manipulator

Case 1 considers pre-processed inputs, i.e. the input joint angle is pre-processed through a cosine function before being fed to the individual trees. In the context of neural networks, Lewis et al. [25,26] showed that inputs pre-processed with system knowledge impart less training burden on the network, and thus result in better overall performance. Whereas, in Case 2, the inputs were fed directly without any pre-processing of the inputs. Hence, a cosine function has been included in the function set to extract the underlying cosine relationships from the dataset.

The average best and average mean results averaged over 10 independent runs are shown in Table 12. The results for Case 1 have shown clearly the excellent performance of HGEP compared with GP.

Also, for Case 2, the results of HGEP are better than simple GP. The convergence characteristics for Cases 1 and 2 are shown in Figs. 3 and 4, respectively. The result in Fig. 3 clearly exhibit the potency of the HGEP method. Moreover, in Fig. 4, for Case 2, the trend of the convergence characteristics of HGEP is constantly improving over the generations, whereas

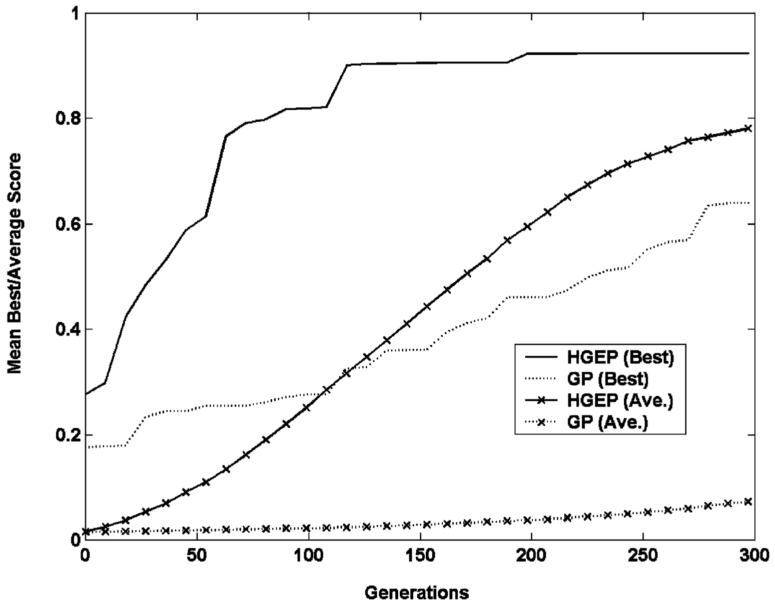

Fig. 3. Average best and mean characteristics of HGEP and GP for one-link manipulator.

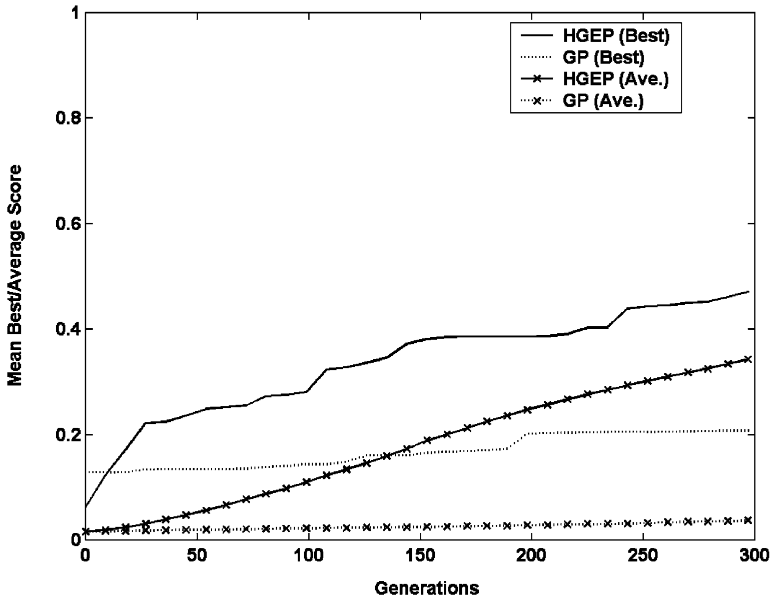

Fig. 4. Average best and mean characteristics of HGEP and GP for one-link manipulator.

for simple GP the trend is almost flat, showing almost no improvement with the advance of generation. The degraded performance in Case 2 clearly shows that automatic model generation with pre-processed inputs always extract better models from the given datasets.

**Case 1.** The best model obtained from the HGEP in Case 1, is given as follows:

$$(+(x(/(y(^*(-0.258717, 0.394407))))))$$

and this can be rewritten as follows:

$$z = x - 9.800098y$$

where $z = \ddot{q}_1$ is the output of the tree individual, $x = \tau_1$, and $y = \cos(q_1)$. As such, in 8 out of the 10 runs, HGEP achieved the best fitness value of one. However, the rest of the best trees with fitness value of one are all structurally distinct from each other.

The best result of the simple GP in all the 10 runs for Case 1 has the fitness of 0.961528, and is given as follows:

$$(+(-(+(-(xy)(/(y - 0.265441))y)(/(y\,0.265441))))$$

This can be simplified to

$$z = x - 2y$$

Clearly, the performance of simple GP is extremely poor.

**Case 2.** On this five-function set, HGEP was able to achieve the targeted best value of one in 1 of the 10 independent runs, and the result is expressed as follows:

$$(-(/(\%(y)) - 0.102035)$$
$$\times(/(x(-(-0.322667, 0.676530)))))$$

This, in its simplified form, can be written as follows:

$$z = 1.00008036x - 9.8005586\cos(y)$$

where $x = \tau_1$, $y = q_1$ and $z = \ddot{q}_1$.

In this case, the best tree of simple GP in all the 10 generations has a fitness of 0.792843, and is expressed as follows:

$$(^*(/(/(\%(y))y)(-(\%(0.466546))$$
$$\times(\%(-0.041987)))y))$$

This, in its simplified notation, is given as follows:

$$z = -9.4347242\cos(y)$$

where $y = q_1$ and $z = \ddot{q}_1$.

Above results show that the performance of simple GP is not good at all. Moreover, it can be seen that a higher value of fitness close to one may not necessarily generate a good model. At least until now, it is clear that a fitness value of one generates a quite accurate model.

### *8.2. Test problem 2: forward dynamics model of two-link robot manipulator*

Here, for all the model description, the terms $x$, $y$, $a$, $b$, $c$, $d$, $e$, $f$, $g$ and $h$ denote $\tau_1$, $\tau_2$, $q_1$, $q_2$, $\dot{q}_1$, $\dot{q}_2$, $\cos(q_1)$, $\cos(q_2)$, $\sin(q_1)$ and $\sin(q_2)$, respectively.

**Case 1.** The evolved tree for $\ddot{q}_1$ is expressed as follows:

$$(A+B+CD)E$$

where

$$A=\frac{g}{f^{13}};$$

$$B=0.504441\left(\frac{a^4}{e^4 f^{11}}+\frac{ea}{0.498535(ga-e)}\right);$$

$$C=f+C'+C''$$

with

$$C'=\frac{a}{f^8}+0.773701\left(\frac{g}{f^5}-\frac{ef^4}{a}\right);$$

$$C''=\frac{a}{f^6}\left\{f-3.893349\left(\frac{a^2-e}{a}\right)\right\}-\frac{ef^9}{a}$$

$$D=e+b; \qquad E=\left(\frac{g}{f^{14}}-\frac{ef^{13}}{g}\right)$$

The second output $\ddot{q}_2$ in its simplified form, can be expressed as $\ddot{q}_2=0.102055e+0.0346979$.

**Case 2.** The evolved tree for $\ddot{q}_1$, in its simplified form, can be expressed as follows:

$$\ddot{q}_1=g-5.121055af - \left\{\left(d-0.039398+d+\frac{xb}{A}\right)+B\right\}C+D$$

where

$$A=\frac{g(e-0.289854)}{xb}+\frac{xb}{c-0.091650}A'+A''$$

with

$$A'=\left(\frac{a}{c}+\frac{xb}{e-0.289854}\right)g;$$

$$A''=c+d(a+b-e-2f+2g-6.388320);$$

$$B=(2a-f)\left\{f+\left(\frac{g}{b+c+0.45697}\right) \times\left(\frac{bg}{eb}+bc+2b+c+0.045697\right)\right\};$$

$$C=\frac{c}{c+g+f+2ceg-1.026757};$$

$$D=2g-f-\frac{0.328311b}{f-bg}+d-c(d+g) \times(2g-f-0.212758)+d +\frac{xbe^2(b+c-0.141237)}{(b+c+xb-141237)g+D'}$$

with

$$D'=e(b+c-0.141237)(c-0.045493)$$

The second output $\ddot{q}_2$, in its simplified form, can be expressed as follows:

$$\ddot{q}_2=ad+2h+2a^2d+(a+0.3599003c -0.6119268)\left(\frac{ad}{c}\right)+ac^3d+2adh -1.153011ad+0.12511$$

**Case 3.** The evolved tree in this case is very large, and in readable form it can be represented as follows:

$$\ddot{q}_1=\frac{(A-BC)D}{(E+F)G}$$

where $A$, $B$, $C$, $D$, $E$, $F$ and $G$ are functions of manipulator parameters and are expressed as

$$A=\frac{eb+d}{x-(b/(ef-y))};$$

$$B=0.4915a\left(g-b+\frac{h}{b}\right)(f-b);$$

$$C=\frac{d+y-0.406856}{0.406856};$$

$$D=\frac{-by(x+2.19286)}{(b/y)+x-y};$$

$$E=\left(-1.40468+\frac{d+by}{0.308786}\right)\frac{e}{b}; \qquad F=\frac{by}{cef};$$

$$G=\frac{d+y-0.406856}{0.406856}$$

For this case, the average score is 0.802978 and the best score is 0.999921. Similar results were noted for $\ddot{q}_2$.

All the experiments on the two-link manipulator were performed with pre-processed inputs. In Case 1, the evolved model is much simpler in structure compared to other two cases. This is mostly because of the use of the data points from a single sinusoid. Moreover, in all the cases, the results do not provide the actual model from which the dataset was generated, although the fitness scores are relatively high. This suggests that the resultant structure in these cases is biased by some dominant modes of the training data, and thus drags the system to that particular mode. It thus appears that the HGEP-based modelling scheme provides a local model rather than a global one. This provision of more clarity to the underlying situation is one of the most important features of GP. In contrast to neural networks, the clear transparent structure of GP is able to provide such typical model characteristics very efficiently. This unique power of GP-based techniques can be utilised as a framework for global data-based modelling. However, in the present context, the results above clearly demonstrate that the HGEP method is not able to find the exact model of a two-link manipulator system. Thus, it is expected that it will also not work for more complex multi-arm manipulator systems.

## 9. Discussion

The results presented in Section 6 can imply that it is necessary for an accurate and precise model generated automatically from a given dataset to have a fitness score that is the same as the original system or the actual model, but it is not a sufficient condition to obtain an exact or a very close approximation of the exact model. This can, otherwise, be stated as that a given model derived from a desired dataset having fitness same as the actual model may not always replicate the actual system.

Thus, an inappropriately generated input–output dataset used for model generation may yield a local model whose fitness is the same as that of the actual model. A local model provides a biased model of the system as in the case of a two-link manipulator, where in Case 1 (Section 6), the model promptly achieves the desired highest level of fitness value. In spite of the high value of the fitness, the model is far from the correct one. This clarity of local and global (actual) model always remains hidden in all non-transparent modelling methods like neural networks. Thus, GP-based methods in general provide a better picture of the system and help in understanding the internal dynamics of the system.

Moreover, during the process of simulation of the automatic generation of the dynamic models, the following important points have been experienced.

- During the learning process, the HGEP method did not include all the designated inputs from the terminal set into the model. Thus, the resultant model becomes biased toward some particular local structure of the system. It is quite obvious that the input–output data collected for training always pertains to some particular task. Hence, the model from this dataset will always be biased towards that task. Thus, in particular, it is quite difficult to obtain a global system model.
- Global models are likely to be obtained from a completely randomly generated input and output set. However, a completely randomly generated dataset with many input and output variables increases the feasible search space enormously. Thus, it is almost impossible to obtain any useful model from a completely random input–output dataset with many inputs. Due to this reason, for a simple system like a single-link manipulator where the number of inputs are very few, the completely random input–output dataset produced the exact model. Whereas, when a two-link manipulator system with many input variables was tested with complete random input–output datasets, the fitness score never exceeded 0.1 with finite population size and learning time. This prohibits any possibility of obtaining good models for a system with many inputs from randomly generated input–output datasets within a limited time with finite population size. Due to this reason, natural sinusoidal type of input variations were chosen to train the HGEP in order to obtain the mathematical model.
- It has been shown that the HGEP method used for data-based intelligent modelling is not capable of reproducing the actual model for even a two-link, rigid

manipulator system. However, it provides greater insight into the internal structure of the problem, compared to other data-based modelling methods, thereby leading to a new dimension in the analysis of intelligent data-based modelling methods. Hence, conventional mathematical models [24–26], although requiring high mathematical skills, are the only way to represent complex robotic manipulator systems. Thus, it is suggested from the results described in this paper that conventional mathematical modelling methods are the only choice at present for the simulation and control operations of complex, non-linear manipulator systems until the development of a highly efficient intelligent data-based algorithm as an automatic model generator.

## 10. Conclusions

This paper has described a hybrid GP and EP (HGEP) method for modelling the task. GP has been used to find an optimal model structure and EP evolves the ephemeral constants contained within a particular GP model structure to make the underlying GP model structure more robust. To speed up the overall process of the model evolution, suitable modifications to the basic EP technique have been performed that use Cauchy distribution instead of the usual normal distribution of BEP. Thus, this yield a very fast EP method, named here as Cauchy-guided EP method. In order to reduce the computational burden, only one tree individual per generation has been chosen to better exploit its underlying structure. Due to the use of a single tree, the CGEP method deals with only one row vector of numerical parameters of the selected tree. Hence, a $(1 + \lambda)$ stochastic selection strategy has been used for updating the numerical parameters of the selected tree individual.

For complex problems, both crossover and mutation operators were used to exploit the program search space. The mutation operation used here consists of a series of mutation operators forming a mutation set. A Poisson variate has been used to select the number of mutation operators to be used for the overall mutation of the selected tree individual. Then, the best model of each generation is picked up, and its ephemeral constants are optimised for best performance. Then, the worst tree in that generation is replaced by the CGEP optimised tree. Thus, in each generation, the best tree of GP coexists with its subsequent optimised supposedly more fit tree.

It has been shown experimentally that, for complex problems, it is always necessary to incorporate system knowledge into the input–output dataset. For a robot manipulator system, pre-processing of the input data by means of the cosine and sine functions greatly improves the overall performance. Hence, it is suggested here that system knowledge should always be incorporated into the input–output dataset to reduce the pressure on the optimisation method, thereby helping the underlying method to yield better results.

It has also been emphasised that, for a proper model, it is necessary for the fitness of the evolved model to be same as the fitness of the actual model, but this is not a sufficient condition to produce an exact system model.

More importantly, it has been shown, whilst that the HGEP method performed better than the conventional tree-coded GP method on many simple model generation tasks including a model for single-link manipulator system, it could not generate the exact forward dynamics model for a two-link manipulator system.

## References

[1] A. Eskandarian, N.E. Bedewi, B.M. Kramer, A.J. Barbera, Dynamic modelling of robotic manipulators using artificial neural network, J. Robotic Syst. 11 (1) (1994) 41–56.

[2] K. Hunt, G. Irwin, K. Warwick, Neural Network Engineering in Dynamic Control Systems, Spinger-Verlag, London, 1995.

[3] O. Omidvar, D.L. Elliott, Neural Systems for Control, Academic Press, New York, 1997.

[4] D. Psaltis, A. Sideris, A.A. Yamamura, A multilayered neural network controller, IEEE Control Systems Magazine, April 1988, pp. 17–21.

[5] P. Marenbach, K.D. Bettenhausen, S. Freyer, U. Nieken, H. Rettenmaier, Data driven structured modelling of a biotechnological fed-batch fermentation by means of genetic programming, in: Proceedings of the Institution of Mechanical Engineers, J. Syst. Control Eng. (1997).

[6] P. Marenbach, M. Brown, Evolutionary versus inductive construction of neuro-fuzzy systems for bioprocess modelling, in: A.M.S. Zalzala (Ed.), Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications GALESIA, University of Strathclyde, Glasgow, UK, 1997, pp. 320–325.

[7] H. Hiden, M. Willis, B. McKay, G. Montague, Non-linear and direction dependent dynamic modelling using genetic programming, in: J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (Eds.), Proceedings of the Second Annual Conference on Genetic Programming, Stanford University, CA, Morgan Kaufmann, San Francisco, 1997, pp. 168–173.
[8] B. McKay, C. Sanderson, M.J. Willis, J. Barford, G. Barton, Evolving a hybrid model of a batch fermentation process, Trans. Inst. Measur. Control (1997).
[9] H. Cao, L. Kang, Z. Michalewicz, Y. Chen, A two-level evolutionary algorithm for modelling system of ordinary differential equations, in: Proceedings of the Third Annual Conference on Genetic Programming, 1998, pp. 17–22.
[10] L.M. Howard, J. D'Angelo, The GA-P: a genetic algorithm and genetic programming hybrid, IEEE Expert, June 1995, pp. 11–15.
[11] O. Castillo, P. Melin, A new-fuzzy–fractal–genetic method for automated mathematical modelling and simulation of robotic dynamic systems, Proc. IEEE Fuzzy Syst. (1998) 1182–1187.
[12] H. Cao, L. Kang, Y. Chen, Evolutionary modelling of ordinary differential equations for dynamic systems, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), 1999, pp. 959–965.
[13] D.E. Goldberg, Genetic Algorithms in Search, Optimisation and Machine Learning, Addison-Wesley, Reading, 1998.
[14] H.P. Schwefel, Numerical Optimisation of Computing Models, Wiley, Chichester, UK, 1981.
[15] D.B. Fogel, Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, IEEE Press, 1995.
[16] L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence Through Simulated Evolution, Wiley, New York, 1966.
[17] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (2) (1999) 82–102.
[18] A.K. Swain, A.S. Morris, Performance Improvement of Self-Adaptive Evolutionary Methods with a Dynamic Lower Bound, Int. J. Inform. Process. Lett. (2001).
[19] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.
[20] J.R. Koza, Genetic Programming: On the Programming of Computers by Natural Selection, MIT Press, Cambridge, MA, 1992.
[21] W. Banzaf, P. Nordin, R.E. Keller, F.D. Francone, Genetic Programming: An Introduction. Morgan Kaufmann, New York, USA, 1998.
[22] A.K. Swain, A.M.S. Zalzala, An overview of genetic programming: current trends and applications, Research report no. 147, University of Sheffield, UK.
[23] N.L. Johnson, S. Kotz, N. Balakrishnan, Continuous Univariate Distributions, vol. 1, Wiley, New York, USA, 1994.
[24] J.J. Craig, Introduction to Robotics: Mechanics and Control, Addison-Wesley, Reading, 1989.
[25] F.L. Lewis, K. Liu, A. Yesildirek, Neural net robot controller with guaranteed tracking performance, IEEE Trans. Neural Networks 6 (3) (1995) 703–715.
[26] F.L. Lewis, S. Jagannathan, A. Yesildirek, Neural Network Control of Robot Manipulators and Nonlinear Systems. Taylor & Francis, London, 1999.

**Anjan Kumar Swain**, received his bachelors of science degree in electrical engineering in 1988, masters of science in engineering in 1991 from Regional Engineering College, Rourkela, India, and PhD degree from the University of Sheffield, UK in 2001. He worked with Electrical Engineering Department of Regional Engineering College Rourkela, India, from 1988 to 1989 and 1991 to 1992. Subsequently, he worked with Ramco Electronics Division, Madras, India from 1992 to 1993 as a real-time process control software engineer. After that he joined as a lecturer in the Electrical Engineering Department of Indira Gandhi Institute of Technology, Orissa, India. He has over 50 publications in journals and conferences. He is the recipient of the national best young teacher award for the year 1996 in India. His current research interests include evolutionary computing methods, evolving networks, dynamics and control of multi-arm robotic manipulator systems. He serves as a reviewer of journals and conferences including IEEE Transactions on System, Man and Cybernetics.

**Alan S. Morris** was educated in UK and graduated with a BEng degree in electrical and electronic engineering from Sheffield University in 1969. After 5 years of employment with British Steel as a research and development engineer in control system applications, he returned to Sheffield University to carry out research in electric arc furnace control, for which he was awarded the degree of PhD in 1978. Since that time, he has been employed as a lecturer, and more recently senior lecturer, at Sheffield, where he now holds the position of Director of Undergraduate Studies. His main research interests lie in robotics, and he is now the author of over 100 refereed research papers. Professionally, he is a Fellow of the Institute of Measurement and Control and a Member of the Institution of Electrical Engineers, and participates as a member of technical panels and organiser of conferences.