EGEP: An Event Tracker Enhanced Gene Expression Programming for Data Driven System Engineering Problems

Zhengwen Huang, *Member, IEEE*, Maozhen Li, Alireza Mousavi, *Senior Member, IEEE*, Morad Danishva, and Zidong Wang, *Fellow, IEEE*

***Abstract*—Gene expression programming (GEP) is a data driven evolutionary technique that is well suited to correlation mining of system components. With the rapid development of industry 4.0, the number of components in a complex industrial system has increased significantly with a high complexity of correlations. As a result, a major challenge in employing GEP to solve system engineering problems lies in computation efficiency of the evolution process. To address this challenge, this paper presents EGEP, an event tracker enhanced GEP, which filters irrelevant system components to ensure the evolution process to converge quickly. Furthermore, we introduce three theorems to mathematically validate the effectiveness of EGEP based on a GEP schema theory. Experiment results also confirm that EGEP outperforms the GEP with a shorter computation time in an evolution.**

***Index Terms*—Gene expression programming, schema theory, event tracker, data driven system engineering.**

I. Introduction

With the rapid development of industry 4.0, the complexity of the modern industrialized manufacture systems has increased significantly. Z-fact0r [33] is a European project aiming to reduce defect rates in manufacturing industry. However, it has become increasingly challenging to mine the correlations of system components in a complex industrial system.

Gene Expression Programming (GEP) [3] is an Evolutionary Algorithm (EA) [4] developed in 2001. It is developed based on the similar idea to Genetic Algorithms (GA) [5] and Genetic Programming (GP) [6]. The linear structure of GA and the tree structure of GP are employed in GEP to operate a genotype-phenotype representation of genetic information which provides a novel representation mechanism for potential candidate solutions. As a result, GEP creates a structured and flexible searching platform for complex problems. GEP has been applied in many fields including combinatorial optimizations [7]–[9] finite transducers [10], classifications [11]–[15], time series predictions [16]–[18] and symbolic regressions [19]–[21].

The flexible structure of GEP together with its black-box style in solution searching makes GEP an outstanding analytic approach to the correlation mining in complex system problem. We have previously applied GEP in particle physics [22]–[24] to discriminate events from the background noisy signals. The performance was further improved with a prefix notation [25] to represent a candidate solution. In another works [26], [27], we applied GEP to mine the correlations of Hadoop [28] parameters for big data analytics. In the Z-fact0r project, we apply GEP to handle data driven system engineering problems.

Based on the state changes of components, GEP generates a mathematical function to represent the correlation of the involved system components. Using GEP, system changes from one state to another are trackable and predictable. However, it should be pointed out that GEP evolution is a computationally intensive process as it considers every component involved in the targeted system.

We have developed an Event Tracker [1] as a Sensitivity Analysis [2] technique for analyzing the contributions of input factors to system state changes. The Event Tracker dynamically tracks the events generated by the components of a complex system and provides an efficient analysis method by eliminating the input factors that have the least impact on the system state changes. The Event Tracker generates a sensitivity index of the input factors in a target system.

The Event Tracker is a computationally efficient technique that focuses on the state changes of the involved system components. It simply takes a snapshot of the system states which helps engineers in observing system performance. However, the Event Tracker does not consider the correlations of the involved system components especially their interactive influences for system performance optimization.

Manuscript received February 26, 2018; revised May 26, 2018 and July 6, 2018; accepted July 20, 2018. Date of current version March 25, 2019. This work was supported in part by the European Union's Horizon 2020 research and innovation program under Grant 723906, in part by the National Basic Research Program (973) of China under Grant 2014CB340404, and in part by the Science and Technology Commission of Shanghai Municipality under Grant 16JC1401300. *(Corresponding author: Zhengwen Huang.)*

Z. Huang, A. Mousavi, and M. Danishva are with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB83PH, U.K. (e-mail: zhengwen.huang@brunel.ac.uk; ali.mousavi@brunel.ac.uk; Morad.danishva@brunel.ac.uk).

M. Li is with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB83PH, U.K., and also with the Key Laboratory of Embedded Systems and Service Computing, Ministry of Education, Tongji University, Shanghai 200092, China (e-mail: maozhen.li@brunel.ac.uk).

Z. Wang is with the Department of Information Systems and Computing, Brunel University London, Uxbridge UB83PH, U.K. (e-mail: zidong.wang@brunel.ac.uk).

Digital Object Identifier 10.1109/TETCI.2018.2864724

In this paper, we present EGEP which employs Event Tracker to efficiently filter out irrelevant input factors. In this way, EGEP relaxes the selection pressure by focusing on relevant input factors that have large impacts on system state changes. As a result, the computation overhead incurred in the evolution process is significantly reduced due to the real-time processing capability of the Event Tracker.

On the other hand, the performance of EGEP is theoretically validated based on the GEP schema theory proposed in our previous work [29]. Schema theory in general describes how EAs work under the pressure of selection [5] and provides a theoretical support for analysis of EAs. By investigating the behaviours and the computational results of the genetic operations, the evolutionary process of an EA can be mathematically described with a set of formulas which are used to represent the propagation of *schemata*.

The major contributions of this paper are as follows:

1) It presents EGEP which combines an Event Tracker with GEP for enhanced computation efficiency in correlation mining in solving data driven system engineering problems.
2) It introduces three theorems to mathematically illustrate how and why EGEP works in a more efficient manner than GEP.

The rest of the paper is organized as follows. Section II reviews related work from the aspects of both event tracking and GEP. Section III briefly introduces the Event Tracker with a focus on the design of EGEP. Section IV introduces three theorems to mathematically validate the performance of EGEP. Section V analyses the experimental results of EGEP in processing a data set on power systems. Section VI concludes the paper and points out future research directions.

II. Related Work

This section reviews related work from the aspects of event tracking and GEP computation speedup.

A. Event Tracking

An event represents a state change of a target system. An event is an encapsulation of data related to all the involved system components and their behavior changes which contribute to system changes from one state to another.

Event tracking focuses on detecting system state changes. It monitors any behavior changes on system components directly. Event tracking provides an efficient mechanism for system engineering analysis by keeping the interested search region only on those components which contribute to a system state change.

The Event Tracker [1] solves complex system engineering problems by mapping the behavior changes detected from system components side and the state changes observed from the whole system side. The meaningful and useful behavior changes of the component side are tracked and recorded to provide decision recommendations for system operators. Those component behavior changes and whole system state changes are further analyzed in order to provide a sensitivity index of the involved components. In our recent work on EventiC [38], events detected from snapshots of the target system running at ideal conditions are further clustered into a lookup table. Although a full track of the target system state changes is not achievable with Event Tracker, some good states are feasible to be repeated with the guide of the lookup table. It is worth noting that EventiC only takes a few seconds to generate events from those snapshots.

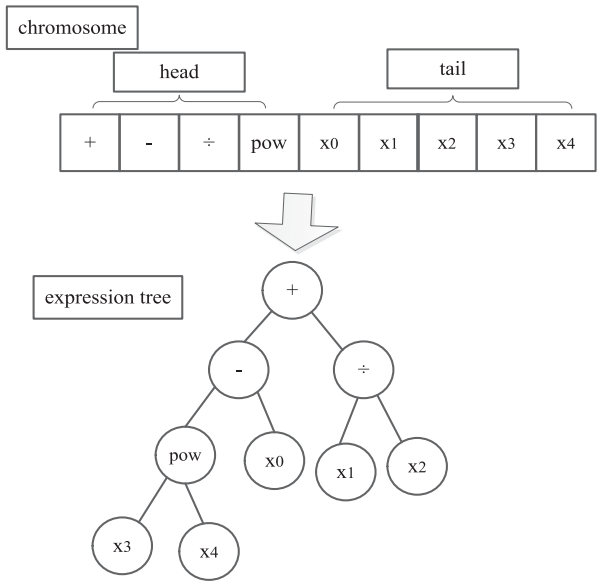


Fig.1. An example of chromosome and expression tree structure.

B. GEP Computation

GEP is a member of the evolutionary algorithms family. It operates on a separated genotype and phenotype mechanism to handle the representation of candidate solutions. The genotype is based on linear element stream structured chromosome and the phenotype is based on Expression Tree (ET) [3] structure.

1) Correlation Representation: A chromosome is the container of genetic information generated from target system. A GEP chromosome can consist of one or more genes. For simplicity in computation, each chromosome has only one gene in this work. A gene is composed of a head and a tail. The head part contains only function elements. These functions are used to describe the correlation among involved factors. The tail part contains function and terminal elements. The terminal elements are used to represent involved system component factors. The length of a gene tail can be computed using (1).

$$Length(Gene_{Tail}) = Length\,(Gene_{Head}) \times (n-1) + 1 \tag{1}$$

Where n is the number of input arguments of a mathematical function which has the most number of input arguments among the functions.

An Expression Tree (ET) is designed to extract and translate information from the chromosome to the solution to target problem. Fig. 1 provides an example which is designed to investigate a system containing 5 input component factors $(x_0, x_1, x_2, x_3, x_4)$. In this example, the size of the gene head is

4 and n is 2. Then the size of the gene tail is 5 based on (1). Four mathematical functions $(+, -, /, pow)$ are selected to represent potential correlations of the parameters x_0, x_1, x_2, x_3, x_4.

As a result of extraction work, a form of $f(x_0, x_1, \ldots, x_n)$ is generated from ET as illustrated in (2).

$$f\ (x_0, x_1, x_2, x_3, x_4) = (pow\,(x_3, x_4) - x_0) + (x_1/x_2) \quad (2)$$

GEP solves a complex system engineering problem with this separated genotype (chromosome) and phenotype mechanism (ET). A linear chromosome structure is employed to maintain complex information generated from a target system. GEP uses ET to extract functions which describe the correlations of the involved components in such a target system. After many generations, the best function maintained by the fittest chromosome of the whole evolution process is generated. Such a function can be used to provide the most accurate description of the contribution of the involved components to a system state change. According to the behavior change of each involved component, the changing mechanism of system states can be further analyzed. Based on the analysis result, the next state of the target system can also be predicted. Therefore, the accuracy of correlation representation in the management level of the search space is achieved.

However, GEP has an evolution mechanism operated by an equally distributed selection pressure. As a major engine of evolution progress, the fitness value only provides a selection pressure to accelerate evolution progress. It does not provide an efficient solution to distinguish whether the involved system components are part of the interested region of search. The reason is that the selection pressure is put on every system component equally. Although GEP can cover all the search space and eliminate irrelevant system components in the later stage of evolution, it may still cause a high computational overhead in processing the space containing irrelevant components. If a filter procedure can be established before starting the main evolution progress, GEP can target at its search in the interesting region.

2) GEP Computation Speedup: Parallelization is a major research direction in GEP computation speedup. A number of parallel GEP solutions have been proposed using a cluster of computers [34]–[36] or a single computer with multiple CPU cores [37]. Recently we have developed P-GEP, a parallel GEP [30] for potential big data analytics. P-GEP follows closely the generation structure of chromosomes in parallelization and considers the input data size in segmentation. P-GEP demonstrates that the input data has a high impact on the computation efficiency of GEP.

Different from these parallel GEP solutions which necessitates powerful computers or computer clusters, EGEP does not require expensive computing resources. It employs an Event Tracker to efficiently filter out irrelevant input factors and provides a more economical way for computation speedup in evolution.

III. EGEP Design and Implementation

This section presents the design of EGEP. Firstly, it briefly introduces the Event Tracker.

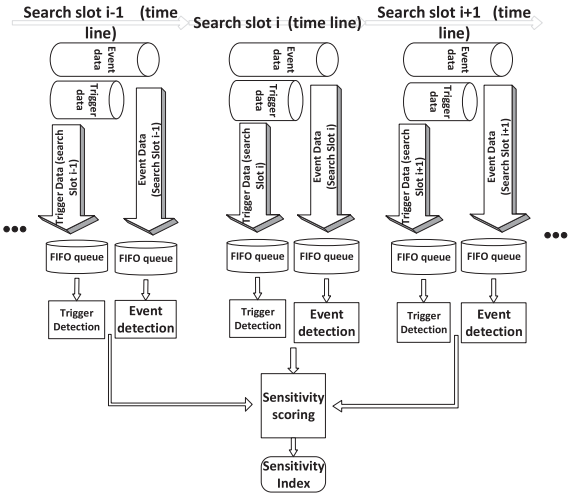


Fig. 2. Event Tracker.

A. Event Tracker

The Event Tracker is employed to reduce the number of involved system components before the main EGEP evolution process starts. The Event Tracker operates four functional parameters:

1) Search Slot - A fixed time slot within which the Event and Trigger data are detected.
2) Analysis Span - A time span within which a period of sensitivity analysis is deployed.
3) Event Threshold - A threshold of fluctuation on event data. It is expressed as a percentage of the range of event data occurring in an analysis span.
4) Trigger Threshold - A threshold of fluctuation on trigger data. It is expressed as a percentage of the range of trigger data occurring in an analysis span.

The integration of the Event Tracker in EGEP is depicted in Fig. 2. In an Analysis Span, a number of Search Slots are set to capture the Event data and Trigger data generated from the target system. In every Search Slot, a pair of Event data and Trigger data are examined with an Event Threshold and a Trigger Threshold respectively. As indicated in (3), if the fluctuation of these values is greater than a pre-defined threshold, there are counted as an Event or Trigger signal.

$$\begin{aligned} &if\ (input_i - input_{i-1}) \geq \theta \overset{Trigger}{\rightarrow} TD_i \\ &if\ (output_i - output_{i-1}) \geq \Psi \overset{Event}{\rightarrow} ED_i \end{aligned} \quad (3)$$

Where, θ and Ψ are the Trigger Threshold value and Event Threshold value respectively. Event and Trigger signals are detected and passed to the sensitivity scoring step. The sensitivity score of a search slot is calculated with a rule that *the simultaneous existence or nonexistence of a change in each pair of data is scored as 1, otherwise the score is −1.*

In an Analysis Span containing n Search Slots, the total sensitivity index score can be calculated with (4).

$$SI_{(t)} = \sum_{1}^{n} search\ slot\ score \quad (4)$$

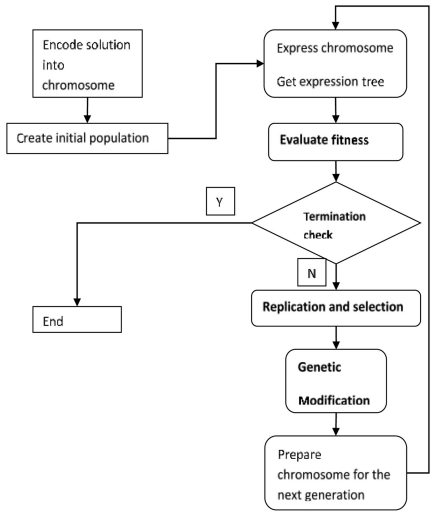


Fig. 3. General GEP evolution process.

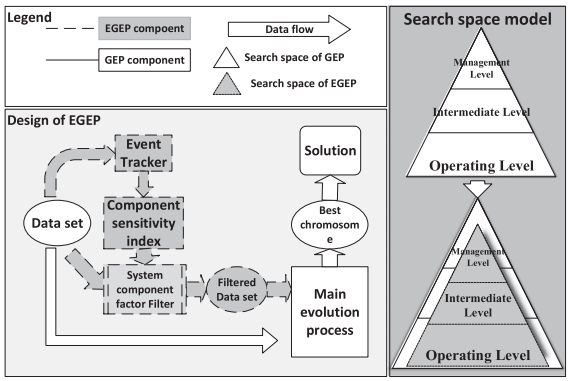


Fig. 4. EGEP and search space model.

where,

1) $SI_{(t)}$ is the sensitivity index score at time t.
2) n is the number of search slots in the analysis span.

In order to compare the sensitivity index score generated from different Analysis Spans, the sensitivity index value is further normalized with (5).

$$\tilde{S} = \frac{SI - l}{u - l} \tag{5}$$

where l and u are the lower and upper bound of index score in its search slot.

By monitoring and tracking the synchronization of behavior changes on system components, Event Tracker generates a sensitivity index which indicates the impacts of system components on an interested system state change. Following such a sensitivity index, the system components which produce low impacts can be removed from the search space of EGEP.

B. EGEP Implementation

EGEP employs Event Tracker to resolve the equally distributed selection pressure problem facing traditional GEPs. Following GEP's evolution structure as shown in Fig. 3, EGEP organizes a more efficient searching space from the operating level to the management level as shown in Fig. 4.

Algorithm 1: GEP Implementation.

Input: A set of running samples of target system;
Output: A correlation of the system component parameters;

```
Initialize the first generation to set best fitness value = 0;
Load original data set;
Apply Event mod part on original data set to generate sensitivity index;
Generate Filtered data set from original data set with sensitivity index;
FOR x = 1 TO restriction degree DO
        exclude the least important factor from filtered data set;
        x++;
ENDFOR;
Load filtered data set;
WHILE i< termination generation number DO
  FOR x = 1 TO size of population DO
  Translate chromosome(x) into expression tree(x);
  Fitness value (x) = Evaluation result of chromosome(x);
  IF fitness value(x) = training sample size THEN
    best chromosome = Chromosome(x) GOTO 29;
  ELSE IF fitness value(x) > best fitness value THEN
    best chromosome = Chromosome(x);
    best fitness value = fitness value(x);
  ENDIF;
  Apply genetic modification on chromosome(x);
  Overwrite the original (x) with the modified chromosome(x);
  x++;
  ENDFOR;
  i++;
ENDWHILE;
Return best chromosome;
```

EGEP keeps the general structure of GEP unchanged in order to maximize the efficiency of the separated genotype and phenotype evolution mechanism. Before starting the main evolution process, EGEP uses Event Tracker to eliminate the irrelevant input factors. In Fig. 4, the dashed parts illustrate how the data is preprocessed by the Event Tracker. The filtered search space is smaller than GEP's original space as shown in Fig. 4 (right part). The detailed EGEP implementation is provided in Algorithm 1.

IV. Theoretical Analysis of EGEP

This section theoretically validates EGEP and presents three theorems to analyze the effectiveness of EGEP based on the schema theory proposed in our previous works [29], [30]. The total execution time T of a GEP evolutionary process can be calculated as

$$T = (T_e \times T_d) \times N_G \tag{6}$$

where

1) T_e is the time to go through the search space in one generation.

2) T_d is the time to process an input data set.
3) N_G is the number of generations.

In this paper we consider the execution time T_{EGEP} of EGEP with three factors which are T_d, T_e and N_G.

Theorem 1: If the number of the involved system components is reduced, EGEP has a shorter input data processing time T_d than GEP.

Proof: In [30], we introduced a theorem to illustrate that a smaller size of an input data set leads to a faster evolutionary process of GEP.

In the system engineering field, the size of an input data set has a high impact on the evolutionary progress. The time in processing an input data set (i.e., T_d) depends on the size of the input data which can be computed as

$$T_d = \sum_{j=1}^{G} \left(\sum_{i=1}^{N_e} (T_{e_i} \times N_d) \right)_j \tag{7}$$

where

1) N_e is the number of elements in a chromosome.
2) G is the number of chromosomes in the current generation.
3) T_{e_i} is the time needed to process the i^{th} element of a chromosome corresponding to a data point in the input data set.
4) N_d is the number of data points in the input data set.

In (7), a data point contains a setting-image sample of the target system. A setting-image contains a snapshot of all the involved system components. Therefore, the data size of this snapshot is directly determined by the complexity of the target system which is defined by the number of components in such a system. The correlation between data size and the number of system components can be expressed with (8):

$$P_s = \sum_{i=1}^{n} CDS_i \tag{8}$$

where

1) P_s is the size of a data point.
2) CDS_i is the size of the i^{th} component in a data point.
3) n is the number of components in a data point.

We further express the execution time of a system engineering problem in GEP with (9)

$$\begin{aligned} T_{sd} &= \sum_{k=1}^{G} \left(\sum_{j=1}^{N_e} \left(T_{e_j} \times N_d \times P_s \right) \right)_k \\ &= \sum_{k=1}^{G} \left(\sum_{j=1}^{N_e} \left(T_{e_j} \times N_d \times \sum_{i=1}^{n} CDS_i \right) \right)_k \end{aligned} \tag{9}$$

As indicated in (9), the time in processing an input data set of a system engineering problem in GEP (T_{sd}) is determined by the number of components contained in a snapshot of target system, n.

Given that

1) A target system X which has two versions of setting snapshots a and b.
2) The number of components in a and b is m and n respectively.
3) $m > n$.

The execution time difference between the two setting snapshots can be calculated with (10) which proves Theorem 1.

$$\begin{aligned} T_{diff} &= T_{sd_a} - T_{sd_b} \\ &= \sum_{k=1}^{G} \left(\sum_{j=1}^{N_e} \left(T_{e_j} \times N_d \times \sum_{i=1}^{m} CDS_i \right) \right)_k \\ &\quad - \sum_{k=1}^{G} \left(\sum_{j=1}^{N_e} \left(T_{e_j} \times N_d \times \sum_{i=1}^{n} CDS_i \right) \right)_k \\ &= \sum_{k=1}^{G} \left(\sum_{j=1}^{N_e} \left(T_{e_j} \times N_d \times \left(\sum_{i=1}^{m} CDS_i \right. \right. \right. \\ &\quad \left. \left. \left. - \sum_{i=1}^{n} CDS_i \right) \right) \right)_k > 0 \end{aligned} \tag{10}$$

Obviously, the formula segment $(\sum_{i=1}^{n} CDS_i - \sum_{i=1}^{m} CDS_i)$ is in the innermost bracket in (10). A small change that happens in this segment leads to a significant change on the T_{diff}.

The Event Tracker reduces the number of the involved system components which increases the value of $(\sum_{i=1}^{n} CDS_i - \sum_{i=1}^{m} CDS_i)$. As a result, EGEP has a shorter time than GEP in processing the input data.

Theorem 2: If the number of the involved system components is reduced, EGEP requires a shorter time T_e to go through the search space in one generation than GEP.

Proof: The time to go through the search space in one generation is determined by the complexity of the search space structure. As mentioned in previous sections, the organization of the search space in GEP is in a hierarchy format which contains three levels, the management level, the intermediate level and the operating level.

Let

1) T_{GEP_e} and T_{EGEP_e} represent the time needed by GEP and EGEP respectively;
2) C_{GEP_O}, C_{GEP_I} and C_{GEP_M} represent the complexity of the operating level, the intermediate level and the management level of GEP respectively;
3) C_{EGEP_O}, C_{EGEP_I} and C_{EGEP_M} represent the complexity of the operating level, the intermediate level and the management level of EGEP respectively;
4) T_U represents the basic unit of processing time.

The operating level is the basement of the whole search space. Due to the pyramid shaped structure as shown in Fig. 4, the size of this level determines the complexity of a search space above it. The total time needed by GEP can be computed with (11)

$$T_{GEP_e} = T_U \times C_{GEP_O} \times C_{GEP_I} \times C_{GEP_M} \tag{11}$$

Since EGEP operates on the same evolution structure as GEP, EGEP shares the similar characteristics with GEP. The T_{EGEP_e} is directly determined by the complexity of the operating level. Using the Event Tracker, the complexity of the operating level is reduced by removing irrelevant system components (the complexity contributed by them is denoted as $C_{GEP_o}{}'$). Compared with C_{GEP_o}, the new complexity of this space can be

calculated with (12)

$$C_{EGEP_o} = C_{GEP_o} - C_{GEP_o}' \tag{12}$$

The contribution from the filtered system components is further accumulated to the up layer of search space. Their contributions are denoted as C_{GEP_I}' and C_{GEP_M}'.

$$\begin{aligned} T_{EGEP_e} &= T_U \times C_{EGEP_o} \times C_{EGEP_I} \times C_{EGEP_M} \\ &= T_U \times (C_{GEP_o} - C_{GEP_o}') \times (C_{GEP_I} - C_{GEP_I}') \\ &\quad \times (C_{GEP_M} - C_{GEP_M}') \end{aligned} \tag{13}$$

Considering (11) and (13) we can generate the difference between EGEP and GEP on T_e using (14) which proves Theorem 2.

$$\begin{aligned} T_{diff} &= T_{GEP_e} - T_{EGEP_e} \\ &= T_U \times \Bigg(C_{GEP_o} \times C_{GEP_I} \times C_{GEP_M} \\ &\quad - C_{EGEP_o} \times C_{EGEP_I} \times C_{EGEP_M} \Bigg) \\ &\geq T_U \times C_{GEP_o} \times C_{GEP_I} \\ &\quad \times (C_{GEP_M} - (C_{GEP_M} - C_{GEP_M}')) > 0 \end{aligned} \tag{14}$$

Theorem 3: If the number of the involved system components is reduced, EGEP requires a smaller number of generations N_G than GEP.

Proof: In GEP, let N_G be the number of generations spent on reaching the best solution in the searching space. The number of generations needed for fully solving a given system engineering problem is depended on the complexity of a search space.

Let

1) N_{GEP_G} and N_{EGEP_G} represents the number of generations needed by GEP and EGEP respectively;
2) $fc(x)$ be a linear increase function which represents the linear increase relation between the complexity of a search space and the number of generations needed.

The N_{GEP_G} and N_{EGEP_G} are expressed with (15) and (16).

$$N_{GEP_G} = fc(C_{GEP_O}, C_{GEP_I}, C_{GEP_M}) \tag{15}$$

$$N_{EGEP_G} = fc(C_{EGEP_O}, C_{EGEP_I}, C_{EGEP_M}) \tag{16}$$

As discussed in the previous section, the operating level is the basement of the whole search space. C_{GEP_I} and C_{GEP_M} are determined by C_{GEP_O}. We further transform (15) and (16) to (17) and (18).

$$N_{GEP_G} = fc(C_{GEP_O}) \tag{17}$$

$$N_{EGEP_G} = fc(C_{EGEP_O}) \tag{18}$$

As mentioned before, the Event Tracker is applied to reduce the complexity of a search space by eliminating some components which have low positions on the sensitivity index. Following the similar solution described in the previous section we can generate the difference between EGEP and GEP on N_G using (19) which proves Theorem 3.

$$\begin{aligned} N_{diff} &= N_{GEP_G} - N_{EGEP_G} \\ &= fc(C_{GEP_O}) - fc(C_{EGEP_O}) > 0 \end{aligned} \tag{19}$$

In this section we have discussed T_d, T_e and N_G for EGEP. They are all smaller than the one required by GEP for solving the same system engineering problem. Following (6) and Theorem 1, Theorem 2 and Theorem 3, we have (20) which validates that theocratically EGEP is faster than GEP in evolution.

$$\begin{aligned} &(T_{EGEP_e} \times T_{EGEP_d}) \times N_{EGEP_G} < (T_e \times T_d) \times N_G \\ &\Rightarrow T_{EGEP} < T_{GEP} \end{aligned} \tag{20}$$

V. Performance Evaluation

In order to evaluate of the performance of EGEP, a set of experiments were designed to track the evolution progress by monitoring the time consumed and the number of generations elapsed to reach specific fitness value(s). Scalability tests were further conducted to analyse the performance of EGEP in dealing with an increasing size of data set. We first introduce the data set employed in the evaluation.

A. Data Set

Power system data set: The total data set contains 9568 data points (measurements) collected from a Combined Cycle Power Plant over 6 years [31], [32]. In this data set, hourly average ambient variables including Temperature (T), Ambient Pressure (AP), Relative Humidity (RH), Exhaust Vacuum (V) etc. are provided to describe the output status change of such power plant. It is divided into two parts, 5000 measurements for training and 4568 measurements for testing. Following our previous work presented in [26], [27], [29], EGEP and GEP are applied to generate a mathematical function which represents the correlation of the power related environmental factors for production prediction of the power plant.

It is worth noting that the data set explored by GEP and EGEP contains information of the two irrelevant noisy components which are intentionally populated with random numbers.

B. Parameter Settings

The settings of EGEP and GEP are listed in Table I. The parameters were set using the classical values used for a traditional GEP.

In order to compare the performance of EGEP and GEP, their execution times and the numbers of generations needed to reach a specific fitness-value threshold on the training data set were observed. Their performance in accuracy was then validated with the testing data set. The stability tests were implemented with a set of duplicated training data sets.

Fitness values is a crucial indicator of the evolution progress. The work presented in [30] shows that the fitness value of the power system data set reaches up to 99.5%. In this work, we divided the evolution process into three progress stages which are the Young Stage, the Middle Stage and the Mature Stage. Taking 99.5% as an upbound value, we set the fitness value

TABLE I
GEP PARAMETER SETTINGS

Parameters	Values	
Population size	100	
No. of genes in a chromosome	1	
Genetic modifications of GEP	one-point recombination rate	30%
	two-point recombination rate	30%
	insertion sequence transposition rate	10%
	inversion rate	10%
	mutation rate	0.44%

TABLE II
FITNESS-VALUES THRESHOLD LIST (ON TRAINING DATA SET)

Fitness slot Id	1	2	3	4	5	6	7
Stage of evolution	Young			Developing		Mature	
Fitness value threshold (%)	60	80	90	96	98	99	99.5

TABLE III
COMPUTING PLATFORM

CPU	Model	Intel Xenon E5-2683
	No. of Cores	32
	No. of Threads	64
Memory	128 GB	
Operation system	Ubuntu 16.04 LTS	
GPU	Nvidia Quadro P6000	

TABLE IV
SETTING OF EVENT TRACKER OF EGEP

Trigger Threshold (%)	0.1,0.5,1,2.5,5
Event Weighting	0.05
Search Samples	5000

threshold of each stage respectively as listed in Table II. These values were set to observe the performance of EGEP in the three levels of a search space which are the operating level, the intermediate level and the management level.

An Intel Xenon Server was configured to implement experiments. The specification is listed in Table III.

The parameters of the Event Tracker were set as shown in Table IV.

C. Sensitivity Results

The Event Tracker in EGEP removes irrelevant components for a target system. We conducted 5 runs with different trigger threshold values to detect and filter the noisy components. Event Tracker found two irrelevant components successfully (the worst sensitivity score generated from the relevant components is significantly higher than the value generated from the populated components). As Table V shows, the irrelevant components are marked with low sensitivity values in all the 5 runs. Therefore, EGEP removed the two irrelevant components from the consideration of the later evolution stage.

TABLE V
SENSITIVITY RESULTS

Trigger Threshold (%)	0.1	0.5	1	2.5	5
Worst sensitivity score of related component	0.0699	0.3311	0.4669	0.8031	0.9623
Noise component 1	0.0164	0.0127	0.0292	0.0373	0.0253
Noise component 2	0.0002	0.004	0.0039	0.0082	0.0328

As shown in Table V, with the value of a trigger threshold increases, the difference between the worst sensitivity score of the system components and the noisy components are increased significantly. This is due to that a system component is more sensitive to the change of the trigger threshold. By tuning the value of a trigger threshold, a high sensitivity score can always be found for a system component but not for a noisy component.

Following the Event Tracker, EGEP selects a range of trigger thresholds and evaluates their sensitivity performance. As a result, those components which always provide low values are detected as noise components.

It is worth noting that the execution times consumed by the sensitivity tests are around 1 to 2 seconds. It takes a small percentage of the whole EGEP execution time. Compared with the time consumed by GEP in processing these irrelevant components, the time spent by Event Tracker can even be ignored.

D. Fitness Function

The distance between the actual production and the estimated production of the power plant is selected as a fitness function. During the evolution progress, the quality of a chromosome is credited with (21).

$$fitness = \begin{cases} 1, \ if(\frac{abs[act-est]}{est} < 0.05) \\ 0, \ if\left(\frac{abs[act-est]}{est} \geq 0.05\right) \end{cases} \tag{21}$$

E. Execution Time Analysis

To evaluate the execution time of EGEP, we conducted 150 runs in total on GEP and EGEP respectively. The execution times (average values of 150 runs) in running the two GEPs to reach every target fitness-value threshold (as shown in Table II) are presented in Table VI. Since the mature stage is a stable period of evolution process, we applied *T-test* [39] on the two groups of results of GEP and EGEP with a fitness threshold of 99% and 99.5% respectively. The results of the *T-test* indicate that EGEP is faster than GEP at a significance level higher than 99.9%. The *T-values* of the two results are 5.867 and 4.926 respectively. Fig. 5 illustrates that the execution times of EGEP largely follow a normal distribution with a fitness value threshold of 99.5%.

TABLE VI
EXECUTION TIME RESULTS

Fitness slot Id	1	2	3	4	5	6	7
Stages of evolution	Young			Developing		Mature	
Fitness value threshold (%)	60	80	90	96	98	99	99.5
Average execution time of GEP(s)	49.27	149.4	440.9	675.9	748.5	832.4	893.5
Average execution time of EGEP(s)	31.34	112.6	308.6	468.5	545.7	573.5	699.4
Computation time saved (s)	17.93	36.8	132.3	207.4	202.8	258.9	194.1

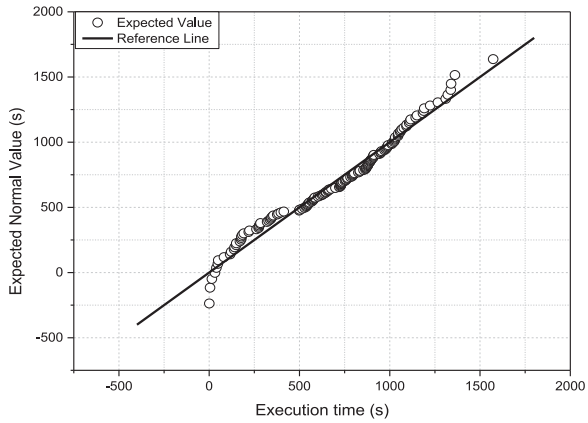


Fig. 5. The distribution of EGEP execution times.

As observed in Table VI, in the Young Stage, EGEP provides a faster evolution speed than GEP. With the Event Tracker, the evolution progress of EGEP is boosted during this period. As a result, in the Developing Stage EGEP achieves a significant leading position. In the Mature Stage, EGEP also enters the stable stage later which leaves EGEP with more space to reach a higher fitness value.

F. Evolution Progress Analysis

The evolution progresses of EGEP and GEP were also tracked to check if EGEP can reach the same fitness-value threshold with a smaller number of generations. According to the fitness-value threshold listed in Table II, we conducted seven groups of experiments. Each group contained 150 executions of the two GEPs respectively. In each execution, the number of generation needed to reach a fitness-value threshold was counted.

As indicated in Table VII, compared with GEP, EGEP always reaches a target fitness-value threshold earlier. It also means that EGEP has a great potential to achieve a higher fitness value using the same number of generations as GEP. The results of the *T-test* indicate that EGEP reaches the fitness value threshold earlier than GEP at a significance level higher than 99.9%. The *T-values* of the two data sets (fitness threshold 99% and 99.5%) are 4.630 and 3.523 respectively.

G. Accuracy Analysis

To validate the performance of EGEP in accuracy, we conducted five groups of experiments. We set 2000, 4000, 6000, 8000 and 10000 as five target generation numbers for those groups respectively. Each group contained 150 executions of EGEP and GEP. When the target generation number was reached the corresponding accuracy of the testing data set was counted. We consider an average value of the 150 executions in each group. The results are presented in Fig. 6 and Table VIII.

As indicated in Fig. 6, EGEP always provides a better average performance in accuracy. Because the Event Tracker filters out irrelevant components (noise factors), EGEP can focus on the relevant system components which speeds up the evolution process. As a result, following the same number of generations, EGEP always generates a more accurate mathematical function than GEP.

TABLE VII
EVOLUTION PROGRESS RESULTS

Fitness slot Id	1	2	3	4	5	6	7
Stages of evolution	Young			Developing		Mature	
Fitness value threshold (%)	60	80	90	96	98	99	99.5
Average number of generation GEP	530	1445	4430	6843	7379	8309	9051
Average number of generation EGEP	400	1329	3620	5301	5916	6361	7826

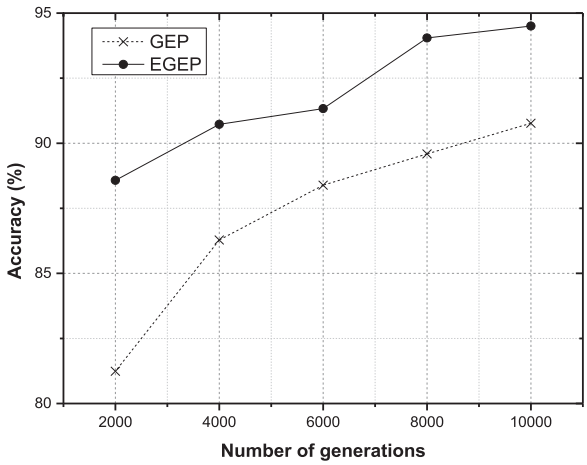


Fig. 6. EGEP performance in accuracy.

TABLE VIII
STANDARD DEVIATIONS OF ACCURACY

No. of Generations	2000	4000	6000	8000	10000
GEP	18.2	12.2	11.8	8.8	8.8
EGEP	11.9	10.5	9.7	7.9	7.2

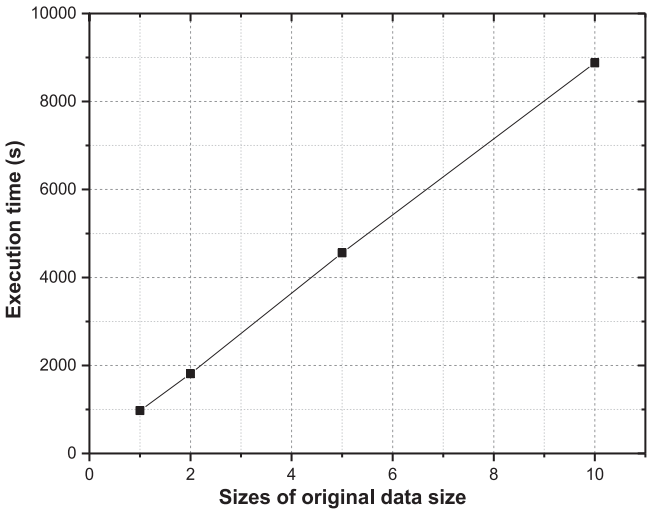


Fig. 7. Scalability of EGEP in computation.

TABLE IX
STANDARD DEVIATIONS OF EGEP IN EXECUTION TIME

Times of original data size	1	2	5	10
Standard deviations of execution time	219.2	352.4	995.5	2020.8

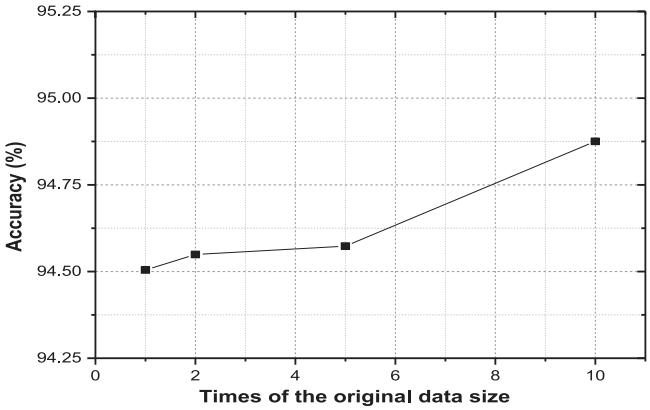


Fig. 8. Scalability of EGEP in accuracy.

TABLE X
STANDARD DEVIATIONS OF EGEP IN ACCURACY

Times of original data size	1	2	5	10
Standard deviations of accuracy	7.2	6.5	7.9	5.7

H. Scalability Analysis

The scalability tests of EGEP were conducted with three duplicated data sets which are 2, 5, 10 times bigger than the original data set. We conducted 30 executions of EGEP on each data set. The performance results in speed and accuracy are shown in Fig. 7, Table IX and Fig. 8, Table X respectively.

As shown in Fig. 7, the linear correlation between execution time and data size confirms a good performance in scalability. Fig. 8 also indicates that EGEP does not lose any accuracy when it is applied on large data sets. It is worth noting that the slight improvement of the accuracy in Fig. 8 is still in the noise width. Part of the improvement is contributed by the larger number of data samples in the duplicated data set. This is because increasing the number of data samples of a population leads to more efficient genetic operations in the evolution [29].

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented EGEP for computation efficiency in solving data driven complex system engineering problems. EGEP employed an Event Tracker to reduce the computation overhead incurred by irrelevant system components during the early stage of evolution. Based on GEP schema theory, we introduced three theorems to mathematically validate the performance of EGEP. Experimental results also confirmed that EGEP has a faster evolution process than GEP.

It should be pointed out that the selection of a trigger threshold remains a challenging issue in the Event Tracker, which relies on a static process. We have further developed EventiC [38] to cluster the events. A future work will research how the clustering results of the EventiC can be utilized to select a trigger threshold dynamically.

REFERENCES

[1] S. Tavakoli, A. Mousavi, and P. Broomhead, "Event tracking for real-time unaware sensitivity analysis (eventtracker)," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 2, pp. 348–359, Feb. 2013.

[2] H. L. Cloke, F. Pappenberger, and J. P. Renaud, "Multi-method global sensitivity analysis (MMGSA) for modelling floodplain hydrological processes," *J. Hydrol. Process.*, vol. 22, pp. 1660–1674, 2008.

[3] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001.

[4] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford Univ. Press, 1996.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[6] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Stat. Comput.*, vol. 4, no. 2, pp. 87–112, 1994.

[7] C. Ferreira, "Combinatorial optimization by gene expression programming: inversion revisited," in *Proc. Argentine Symp. Artif. Intell.*, 2002, pp. 160–174.

[8] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multi-armed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 217–228, Feb. 2015.

[9] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 309–325, Jun. 2015.

[10] J. S. Manognya and L. P. Wang, "Gene expression programming for induction of finite transducer," in *Proc. 7th Int. Conf. Inf. Commun. Signal Process.*, 2009, pp. 1–5.

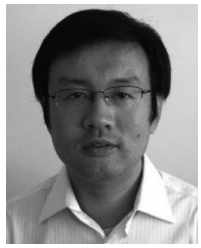
[11] C. Ferreira, "Discovery of the boolean functions to the best density-classification rules using gene expression programming," in *Proc. 5th Eur. Conf. Genetic Program.*, 2002, vol. 2278, pp. 50–59.

[12] C. Zhou, P. C. Nelson, W. Xiao, and T. M. Tirpak, "Discovery of classification rules by using gene expression programming," in *Proc. Int. Conf. Artif. Intell.*, Jun. 2002, pp. 1355–1361.

[13] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Trans. Evol. Comput.*, vol. 7, no. 6, pp. 519–531, Dec. 2003.

[14] M. H. Marghny and I. E. El-Semman, "Extracting logical classification rules with gene expression programming: Microarray case study," in *Proc. Int. Conf. Artif. Intell. Mach. Learn.,* Cairo, Egypt, 2005, pp. 11–16.

[15] S. W. Wilson, "Classifier conditions using gene expression programming," in *Proc. Int. Workshop Learn. Classifier Syst.*, 2008, vol. 4998, pp. 206–217.
[16] J. Zuo, C. Tang, C. Li, C. Yuan, and A. Chen, "Time series prediction based on gene expression programming," in *Proc. 5th Int. Conf. Adv. Web-Age Inf. Manage.*, 2004, vol. 3129, pp. 55–64.
[17] V. I. Litvinenko, P. I. Bidyuk, J. N. Bardachov, V. G. Sherstjuk, and A. A. Fefelov, "Combining clonal selection algorithm and gene expression programming for time series prediction," in *Proc. 3rd Workshop IEEE Intell. Data Acquis. Adv. Comput. Syst. Technol. Appl.*, 2005, pp. 133–138.
[18] H. S. Lopes and W. R. Weinert, "A gene expression programming system for time series modeling," in *Proc. XXV Iberian Latin Amer. Congr. Comput. Methods Eng., Recife*, vol. 11, 2004, pp. 10–12.
[19] H. S. Lopes and W. R. Weinert, "An enhanced gene expression programming approach for symbolic regression problems," *Int. J. Appl. Math. Comput. Sci.*, vol. 14, no. 3, pp. 375–384, 2004.
[20] Z. Cai *et al.*, "Symbolic regression based on GEP and its application in predicting amount of gas emitted from coal face," in *Proc. Int. Symp. Saf. Sci . Technol.*, vol. 637641, 2004.
[21] E. Bautu, A. Bautu, and H. Luchian, "Symbolic regression on noisy data with genetic and gene expression programming," in *Proc. 7th Int. Symp. Symbolic Numer. Algorithms Sci. Comput.*, 2005, pp. 321–324.
[22] L. Teodorescu and Z. Huang, "Enhanced gene expression programming for signal-background discrimination in particle physics," in *Proc. XII Adv. Comput. Anal. Techn. Phys. Res.*, vol. 70, 2008, p. 66.
[23] L. Teodorescu, "Gene expression programming approach to event selection in high energy physics," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 4, pp. 2221–2227, Aug. 2006.
[24] L. Teodorescu, "High energy physics data analysis with gene expression programming," in *Proc. Int. Symp. IEEE Nucl. Sci.*, 2005, vol. 1, pp. 143–147.
[25] X. Li, C. Zhou, W. Xiao, and P. C. Nelson, "Prefix gene expression programming," in *Proc. Int. Conf. Genetic Evol. Comput.*, 2005, pp. 25–29.
[26] M. Khan, Z. Huang, M. Li, G. A. Taylor, and M. Khan, "Optimizing hadoop parameter settings with gene expression programming guided PSO," *Concurrency Comput. Pract. Exp.*, vol. 29, no. 3, 2016, Art. no. e3786. doi: 10.1002/cpe.3786.
[27] M. Khan, Z. Huang, M. Li, G. A. Taylor, P. M. Ashton, and M. Khan, "Optimizing hadoop performance for big data analytics in smart grid," *Math. Probl. Eng.*, vol. 2017, 2017, Art. no. 2198262.
[28] Apache Hadoop, 2011. [Online]. Available: https://hadoop.apache.org/. Accessed on: Feb. 11, 2018.
[29] Z. Huang, "Schema theory for gene expression programming," Ph.D. dissertation, Brunel University London, Uxbridge, U.K., 2014.
[30] Z. Huang, M. Li, C. Chousidis, A. Mousavi, and C. Jiang, "Schema theory based data engineering in gene expression programming for big data analytics," *IEEE Trans. Evol. Comput.*, to be published, doi: 10.1109/TEVC.2017.2771445.
[31] P. Tüfekci, "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *Int. J. Electr. Power Energy Syst.*, vol. 60, pp. 126–140, 2014.
[32] H. Kaya, P. Tüfekci, and S. F. Gürgen, "Local and global learning methods for predicting power of a combined gas & steam turbine," in *Proc. Int. Conf. Emerg. Trends Comput. Electron. Eng.*, 2012, pp. 13–18.
[33] Z-fact0r, 2011. [Online]. Available: http://www.z-fact0r.eu/. Accessed on: Feb. 11, 2018.
[34] Z. Cai *et al.*, "A novel algorithm of gene expression programming based on simulated annealing," in *Proc. Int. Symp. Intell. Comput. Appl.*, vol. 610, 2005
[35] M. Snir, S. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, "MPI: The Complete Reference," *Scientific Eng. Comput.*, vol. 2. 1996.
[36] X. Du, L. Ding, and L. Jia, "Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm," in *Proc. 4th Int. Conf. Nat. Comput.*, 2008, pp. 433–437.
[37] W. Jiang, T. Li, B. Fang, Y. Jiang, Z. Li, and Y. Liu, "Parallel niche gene expression programming based on general multi-core processor," in *Proc. Int. Conf. Artif. Intell. Comput. Intell.*, 2010, vol. 3, pp. 75–79.
[38] M. Danishvar, A. Mousavi, and P. Broomhead, "EventiC: A real-time unbiased event-based learning technique for complex systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: 10.1109/TSMC.2017.2775666.
[39] J. F. Box, "Guinness, gosset, fisher, and small samples," *Stat. Sci.*, vol. 2, no. 1, pp. 45–52, 1987.



Zhengwen Huang (M'18) received the M.Sc. degree from the King's College London, London, U.K., and the Ph.D. degree from the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K., in July 2014. His research interests include evolutionary algorithms (gene expression programming, genetic programming) and data engineering.

Maozhen Li received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 1997. He was a Postdoctoral Research Fellow with the School of Computer Science and Informatics, Cardiff University, U.K., during 1999–2002. He is currently working as a Professor with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K. His research interests include the areas of high performance computing, big data analytics, and intelligent systems. He is on the Editorial Boards of a number of journals. He has more than 150 research publications in these areas. He is a Fellow of the British Computer Society and the Institute of Engineering and Technology.

Alireza Mousavi (SM'13) received the Ph.D. degree from the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K., in 2000. He is currently working as a Reader of systems engineering and computing. His research interests include smart supervisory control and data acquisition systems applied to real-time systems modeling and optimization. The key areas of application are in stochastic modeling, ontology alignment, and sensor networks.

Zidong Wang (F'14) was born in Jiangsu, China, in 1966. He received the B.Sc. degree in mathematics from Suzhou University, Suzhou, China, in 1986, and the M.Sc. degree in applied mathematics and the Ph.D. degree in electrical engineering both from the Nanjing University of Science and Technology, Nanjing, China, in 1990 and 1994, respectively. He is currently working as a Professor of dynamical systems and computing with the Department of Computer Science, Brunel University London, Uxbridge, U.K. From 1990 to 2002, he held teaching and research appointments with universities in China, Germany, and the U.K. He has published more than 300 papers in refereed international journals. His current research interests include dynamical systems, signal processing, bioinformatics, and control theory and applications. Dr. Wang was a recipient of the Alexander von Humboldt Research Fellowship of Germany, the JSPS Research Fellowship of Japan, and the William Mong Visiting Research Fellowship of Hong Kong. He serves (or has served) as the Editor-in-Chief for neurocomputing and an Associate Editor for 12 international journals, including the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, the IEEE TRANSACTIONS ON NEURAL NETWORKS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C. He is a fellow of the Royal Statistical Society and a member of program committee for many international conferences.

Morad Danishva received the Ph.D. degree from Brunel University London, Uxbridge, U.K., in 2015. He is currently working as a Research Fellow with the System Engineering Research Group, Brunel University London, Uxbridge. His research interests include real-time systems modeling, middlewares, and data engineering. He currently works on the EU Horizon 2020 Z-Factor project, which focuses on zero defects in manufacturing systems. He is a member of IET.