Schema Theory-Based Data Engineering in Gene Expression Programming for Big Data Analytics

Zhengwen Huang, Maozhen Li, Christos Chousidis, Alireza Mousavi, and Changjun Jiang

***Abstract*—Gene expression programming (GEP) is a data driven evolutionary technique that well suits for correlation mining. Parallel GEPs are proposed to speed up the evolution process using a cluster of computers or a computer with multiple CPU cores. However, the generation structure of chromosomes and the size of input data are two issues that tend to be neglected when speeding up GEP in evolution. To fill the research gap, this paper proposes three guiding principles to elaborate the computation nature of GEP in evolution based on an analysis of GEP schema theory. As a result, a novel data engineered GEP is developed which follows closely the generation structure of chromosomes in parallelization and considers the input data size in segmentation. Experimental results on two data sets with complementary features show that the data engineered GEP speeds up the evolution process significantly without loss of accuracy in data correlation mining. Based on the experimental tests, a computation model of the data engineered GEP is further developed to demonstrate its high scalability in dealing with potential big data using a large number of CPU cores.**

***Index Terms*—Big data analytics, data engineering, gene expression programming (GEP), parallelization and segmentation, schema theory.**

I. Introduction

GENE expression programming (GEP) [1] is a member of evolutionary algorithms (EAs) [2] with a similar idea to both genetic algorithms (GAs) [3] and genetic programming (GP) [4]. GEP operates on a genotype–phenotype system to handle the representation of a candidate solution. GEP combines the linear structure of GA with the tree structure of GP providing a structured and flexible mechanism in searching for solutions.

GEP has been applied to many problems, including combinatorial optimizations [6], finite transducers [42], classifications [7]–[10], [41], time series predictions [11]–[13], and symbolic regressions [14]–[16]. GEP was also employed to automatically generate a hyper-heuristic framework for combinatorial optimization problems [43], [44].

We have previously applied GEP in particle physics [17]–[19] to discriminate events from the background noisy signals. The performance was further improved with a prefix notation [20] to represent a candidate solution. In another work [39], we applied GEP to mine the correlations of Hadoop [40] parameters for big data analytics. GEP also has many applications in power systems, such as the short-term load forecasting problem [21] and the static security problem [22].

The flexible structure of GEP together with its black-box style in solution searching makes GEP an appealing analytic approach to big data problems. However, the sheer size of big data would put a heavy burden on GEP computation in evolution. To speed up this process, a number of parallel GEP algorithms have been proposed using a cluster of computers [24], [26] or a single computer with multiple CPU cores [27]. Although the execution time of GEP decreases with an increasing number of CPU processors, these parallel GEPs suffer from two major limitations. On one hand, these parallel GEPs simply distribute the computation of chromosomes across a number of CPUs which breaks the generation structure of GEP leading to inefficiency in evolution. For example, the work presented in [26] assigns CPUs to process the chromosomes simultaneously, but it does not guarantee that the chromosomes of the same generation are assessed together in one iteration. On the other hand, these GEPs have not considered the size of an input data in parallelization leading to a scalability issue when dealing with an ever-growing size of potential big data. Therefore, the generation structure of chromosomes and the size of input data are two issues that tend to be neglected when speeding up the evolution process of GEP.

To fill the research gap, this paper presents a novel data engineered GEP and makes four major contributions.

1) It proposes three guiding principles to elaborate the computation nature of GEP in evolution, which provides a theoretical foundation for GEP parallelization and segmentation. This is based on an analysis of our

Manuscript received December 14, 2016; revised March 15, 2017, May 24, 2017, and October 31, 2017; accepted November 3, 2017. Date of publication December 12, 2017; date of current version September 28, 2018. This work is supported in part by the National Fundamental Research Program (973) of China under Grant 2014CB340404, and in part by the Science and Technology Commission of Shanghai Municipality under Grant 16JC1401300. This research has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723906. *(Corresponding author: Maozhen Li.)*

Z. Huang and A. Mousavi are with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB8 3PH, U.K. (e-mail: zhengwen.huang@brunel.ac.uk; ali.mousavi@brunel.ac.uk).

M. Li is with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB8 3PH, 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).

C. Chousidis is with the School of Computing and Engineering, University of West London, Slough W5 5RF, U.K. (e-mail: christos.chousidis@uwl.ac.uk).

C. Jiang is with the Department of Computer Science and Technology, Tongji University, Shanghai 200092, China (e-mail: cjjiang@tongji.edu.cn).

Digital Object Identifier 10.1109/TEVC.2017.2771445

previous work on GEP schema theory [23] which is also highlighted by Zhong *et al.* [38] in their work.

2) Different from the existing GEP solutions, the data engineered GEP follows closely the generation structure of chromosomes leading to an efficient process in evolution.
3) It employs two segmentation schemes to further speed up the evolution process. The *cutting-in-sequence* scheme segments an input data set into a number of overlapped data chunks with an aim to maintain the accuracy level of GEP in processing segmented data. The *random selection* scheme selects samples from an input data set without overlapping and builds a single data chunk for processing.
4) A computation model of the data engineered GEP is developed to demonstrate its high scalability in dealing with potential big data.

The data engineered GEP is evaluated on two data sets with complementary features. One data set has complex but loosely coupled data samples in that each sample has a large number of input factors. The other data set has strongly correlated data samples but each sample has a small number of input factors. Experimental results show that the data engineered GEP reduces the computation time significantly without loss of accuracy in processing the segmented data chunks, which makes it scalable in dealing with potential big data problems.

The rest of this paper is organized as follows. Section II gives a review on related work. Section III proposes three guiding principles to elaborate the computation process of GEP based on an analysis of GEP schema theory. Section IV details the implementation of the data engineered GEP from the aspects of segmentation, overlapping, and parallelization. Section V evaluates the performance of the data engineered GEP. Section VI develops a computation model to further demonstrate the scalability of the data engineered GEP in dealing with potential big data settings. Section VII concludes this paper and points out some future work.

II. Related Work

The majority of existing works on data engineering in GEP only focus on parallelization. This section reviews some of the representative works in this aspect. It first reviews some works on schema theory which provides a theoretical foundation for GEP computation analysis.

A. Schema Theory

Schema theory is used to describe how EAs work under the pressure of selection. A solution provided by EAs can be considered as a point in a search space, which contains all the possible solutions to a problem. The *schemata* of a chromosome containing such a solution can be considered as the coordinates of the point in the search space. In order to find the location of a good solution, a guided search space is provided by the *schemata* of a chromosome during the evolutionary process [3]. The *schemata* are generated by linking a set of schema elements based on the output of a fitness function. In this way, the search space containing a good solution is explored point by point in the search space and eventually the best solution can be generated.

Schema theory provides a theoretical support for analysis of EAs. By investigating the behaviors and the execution results of the genetic operations, the evolutionary process of EAs can be mathematically described with a set of formulas which are used to represent the propagation of *schemata*.

Holland [3] developed a GA schema theory to explain the evolutionary mechanism of GA. The theorem predicts the number of strings matching a schema in the next generation based on the genetic information of the current generation. Following Holland's GA schema theory, Koza [28] made the first attempt to define the schema in GP as a subspace containing a set of subtrees which share similar output behaviors. The GP schema is a tree structure which provides a deeper understanding of the input data. Poli and Langdon [30] introduced a *fixed-size-and-shape* schema which provides more restrictions on the shape of the *S-expression* program matching the schema. *S-expression* is a data representation of nested lists. In a later version, Poli and McPhee [31], [32] developed a Cartesian node reference system to enhance the positional connection between the schema and the tree structure. Each position in the tree structure is indexed with one point in the node reference system. As a result, a more precise analysis of the propagation of the tree fragments matching the schema can be obtained. All these works try to provide a structured and flexible mechanism for a clear understanding of the GP evolutionary process.

GEP is a relatively new EA algorithm. As a result, few studies have been proposed on GEP schema theory. Cheng and Xue [29] attempted to define GEP schema following closely the work on GA schema theory. This paper does not fully consider GEP specific features, such as the head–tail structure of a chromosome and the phenotype–genotype translation mechanism.

Huang [23] proposed a GEP schema theory which takes into account the GEP specific features in a systematic way. This paper defines a schema together with a set of corresponding theorems to predict the propagation of a schema from one generation to another taking into account the head–tail structure of a chromosome. The phenotype–genotype separation is also considered. The genotype is used to select a schema which can be part of an entire chromosome, not only the part of the open reading frame [1]. The phenotype is used only to provide the natural selection pressure through the fitness values of the chromosomes containing a schema.

Recently, Zhong *et al.* [38] proposed a self-learning GEP in which each chromosome is embedded with subfunctions that can be deployed to construct the final solution. It is worth noting that this paper can be theoretically explained by the schema theory proposed in our previous work [23]. The evolutionary process is actually conducted by accumulating the genetic information on *schemata* which can be computed mathematically. As a result, the proposed self-learning GEP provides a mechanism to maintain the structure of the accumulated *schemata* which leads to an enhanced performance.

B. Parallel GEP

There are a number of works in parallelization of GEP using a cluster of computers. For example, Zhihua *et al.* [24] proposed a hybrid parallel GEP combined with simulated annulling (HPGEPSA) using MPI [25] to achieve parallelism. In HPGEPSA, a new generation only can be generated when all the participating computers finish their computations. As a result, the computation improvement through parallelization is not significant especially when different types of CPU processors are used with varied computing powers.

The asynchronous distribute parallel GEP based on the estimation of distribution algorithm (ADPGEPEDA) further optimizes the load of each participating processor [26] using MPI. In ADPGEPEDA, each computer controls the evolutionary process of a part of the population independently. Since the computation capability of each participating computer is considered, ADPGEPEDA performs better than HPGEPSA in parallelization. However, the evolutionary process in ADPGEPEDA does not guarantee the chromosomes of the same generation would be assessed together in an evolutionary iteration which might break the nature of the selection process leading to an inefficient evolution.

Wu *et al.* [27] presented a parallel NICHE GEP (PNGEPMP) using a single computer with multiple CPU cores for parallelization. Since there is no delay in computation among the homogeneous CPU processors, PNGEPMP achieves an impressive speedup in computation compared with ADPGEPEDA. However, PNGEPMP only focuses on covering more points in the search space by calculating the best fitness value generated from part of a chromosome, which does not represent the behavior of the whole chromosome. As a result, the accumulation of genetic information is not properly maintained in PNGEPMP.

Summarising, the aforementioned parallel implementations only focus on parallelization of the computation of GEP, but do not follow closely the generation nature of GEP leading to inefficiency in evolution. Furthermore, to make a parallel GEP scalable in dealing with potential big data, data engineering techniques, such as segmentation should also be considered.

III. GEP Schema and Computation

In this section, we present three guiding principles to elaborate the computation nature of GEP. First, we briefly describe how the genotype is translated into the phenotype in GEP and how the selection is conducted.

A. Genotype–Phenotype Translation

GEP combines a linear structured genotype chromosome with a phenotype expression tree (ET) [1] as shown in Fig. 1. In this example, the targeted problem has four input parameters (a, b, c, and d) and three mathematic function operators {"+," "− ," "∗"}. The chromosome has only one gene which is composed of a head and a tail. The elements of the head are selected randomly from both the input parameters and the mathematic function operators. The elements of the tail are selected randomly only from the input parameters.

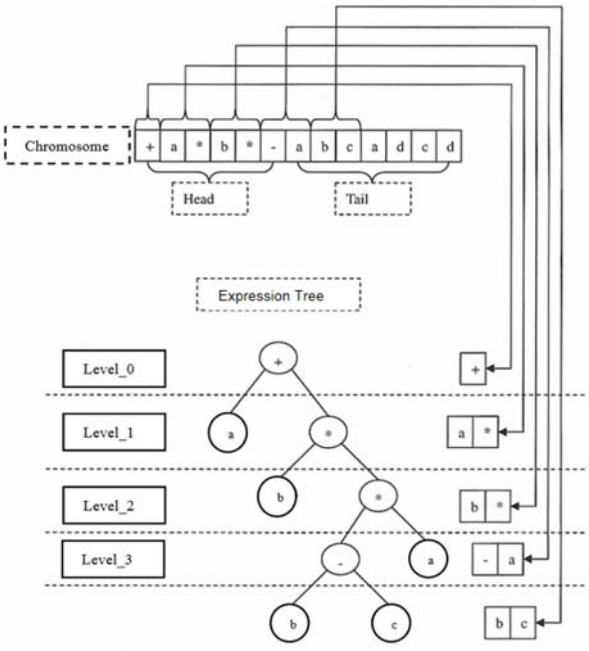


Fig. 1. Example of translation from a chromosome to ET.

The number of the elements of a gene is fixed which can be defined by the user. The relation between the length of the head and the length of the tail can be calculated as

$$\text{Tail} = \text{Head} * (n-1) + 1 \tag{1}$$

where n is the maximum number of arguments that a function operator requires.

The chromosome combines the input parameters and function operators during the evolutionary process. The ET is used to express the correlations among the input parameters. In this example, a candidate correlation among these parameters is represented with a combination of the function operators, i.e., $(a + b * ((b - c) * a))$. The translation from genotype to phenotype in GEP is conducted in the following steps.

1) The element in the chromosome containing the function of $+$ is selected to build the root of ET.
2) The input parameter a and the function * are selected to be placed on Level_1 as the leaf nodes of the function of $+$ in the ET.
3) For the function of * in Level_1, another two elements (input parameter b and function *) are selected to be placed on Level_2 as the leaf nodes of the Level_1 function of * in the ET.
4) The translation process continues until the ET is fully filled with the input parameters.

It is noted that not all the elements in the tail are involved in the translation process which is a typical feature of GEP (i.e., open reading frame [1]). Based on their fitness values, chromosomes in GEP are selected proportionally in evolving into the next generation.

B. GEP Computation Analysis

Following the GEP schema theory proposed in [23], the computation time of GEP in evolution consists of two parts.

One part is related to the search space starting from a chromosome of the initial generation to the best chromosome of the last generation. The other part in computation is related to the size of an input data set. The total execution time T of a GEP evolutionary process can be calculated as

$$T = (T_e \times T_d) \times N_G \tag{2}$$

where

T_e time to go through the search space in one generation;
T_d time to process an input data set;
N_G number of generations.

The evolution of GEP is actually a process in which some segments of a chromosome are found useful and linked together to build the best chromosome. Considering the performance of the chromosomes that have similar genetic characteristics in the current generation, the schema theory [23] estimates the number of the chromosomes with such characteristics in the next generation. A schema is defined as a segment of a chromosome and maintains a certain amount of genetic information. In turn, a chromosome consists of a number of *schemata* representing all the possible solutions in a search space. The search space is created with the feature dimensions of an input data set and a chromosome which provides a structure to maintain the feature dimensions in the coordinate space.

The search space will be traversed during the evolutionary process to generate a number of *schemata* which are linked within a chromosome. The genetic information which is learned from the input data set is also accumulated by linking the *schemata*. At the end of the evolutionary process, the best chromosome which consists of the linked *schemata* is generated to represent the final solution to a targeted problem.

Based on the above analysis of the schema theory, we now propose three guiding principles to elaborate the computation nature of GEP evolution.

1) Guiding Principle 1: To efficiently accumulate the genetic information, the chromosomes of the same generation must be processed together in one evolutionary iteration.

a) Supporting arguments: As indicated in the GEP schema theory, the evolutionary process is an accumulation of genetic information which is maintained in a chromosome. Schema is a segment of a chromosome which contains genetic information useful for a solution. The evolutionary process that a schema is propagated into the next generation can be represented by

$$E[M(H, t+1)] = M \times P_R(H, t) \times P_{\mathrm{GM}}(H, t) \tag{3}$$

where

H schema;
t number of generations;
M number of chromosomes in a generation;
$M(H, t+1)$ number of chromosomes matching the schema H in the generation of $t+1$;
$E[M(H, t+1)]$ estimation of $M(H, t+1)$;
$P_R(H, t)$ probability of a chromosome that matches H and is selected for replication taking into account all the chromosomes in the generation t;
$P_{\mathrm{GM}}(H, t)$ probability that the schema H is still valid after the genetic modification process taking into account all the chromosomes in the generation t;
$M \times P_R(H, t) \times P_{\mathrm{GM}}(H, t)$ theoretical number of chromosomes matching the schema H in the generation of $t+1$.

The evolution progresses with an increasing number of chromosomes that match the schema H from one generation to the next generation. $P_R(H, t)$ relies on the genetic operations which are performed on the chromosomes. A genetic operation is performed on all the chromosomes of the same generation with an aim to maximize the exchange of genetic information among these chromosomes. $P_R(H, t)$ can be calculated by

$$P_R(H, t) = M(H, t) \times \frac{\bar{f}(H, t)}{M \times \bar{f}(t)} \tag{4}$$

where

$M(H, t)$ number of the chromosomes matching H in the generation of t;
$\bar{f}(H, t)$ average fitness value of the chromosomes matching H in the generation of t;
$\bar{f}(t)$ average fitness value of all the chromosomes in the generation of t.

Let $P_R{}'(H, t)$ represent the probability of a chromosome that matches H and is selected for replication taking into account only a group of the chromosomes in a generation. We have

$$\begin{aligned} P_R{}'(H, t) &= \sum_{i=1}^{n} \left(P_{R_i}(H, t) \times \frac{m_i}{M} \right) \\ &= \sum_{i=1}^{n} \left(m_i(H, t) \times \frac{\bar{f}_i(H, t)}{m_i \times \bar{f}_i(t)} \times \frac{m_i}{M} \right) \\ &= \sum_{i=1}^{n} \left(\frac{F_i(H, t)}{F_i(t)} \times \frac{m_i}{M} \right) \\ &\le \sum_{i=1}^{n} \frac{F_i(H, t)}{F_i(t)} \times \frac{M}{M} = \sum_{i=1}^{n} \frac{F_i(H, t)}{F_i(t)} \\ &\le M(H, t) \times \frac{\bar{f}(H, t)}{M \times \bar{f}(t)} \end{aligned} \tag{5}$$

where

$P_{R_i}(H, t)$ probability of a chromosome matching H that is selected from the ith group of the chromosomes in the generation of t;
n number of groups of the chromosomes in the generation of t;
m_i number of chromosomes in the ith group;
$F_i(H, t)$ sum of the fitness values of the chromosomes matching H in the ith group of the generation of t;
$F_i(t)$ sum of the fitness values of all the chromosomes in the ith group of the generation of t.

Considering (4) and (5), we have

$$P_R{}'(H, t) \le P_R(H, t). \tag{6}$$

We denote $P_{\mathrm{GM}}{}'(H, t)$ as the probability that the schema H is still valid after the genetic modification process considering

only a group of the chromosomes in a generation. Following the deduction process of (5), we have

$$P_{\mathrm{GM}}{}'(H,t) \le P_{\mathrm{GM}}(H,t). \tag{7}$$

Let $E[M(H,t+1)]'$ represent an estimation of the number of chromosomes that match the schema H considering only a group of chromosomes in the generation of t. Based on (6) and (7), we have

$$E[M[H,t+1]] \ge E[M[H,t+1]]' \tag{8}$$

which indicates that a group of chromosomes matching schema H in a generation would lead to an evolutionary progress not faster than the case when all the chromosomes in the same generation are processed together.

2) Guiding Principle 2: A smaller size of an input data set leads to a faster evolutionary process of GEP.

a) Supporting argument: The size of an input data set has an impact on the evolutionary progress of GEP. As indicated in (2), 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

$$T_d = \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(T_{e_i}\times N_d\right)\right)_j \tag{9}$$

where

- N_e number of elements in a chromosome;
- G number of chromosomes in the current generation;
- T_{e_i} time needed to process the ith element of a chromosome corresponding to a data point in the input data set;
- N_d number of data points in the input data set.

We denote $T_d{}'$ as the execution time to process a data chunk which is smaller than the original input data set. $T_d{}'$ can be computed as

$$T_d{}' = \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(T_{e_i}\times N_d{}'\right)\right)_j \tag{10}$$

where $N_d{}'$ is the number of data points in a data chunk.

Based on (9) and (10), the execution time difference between the original input data set and a segmented data chunk can be computed as

$$\begin{aligned} T_d - T_d{}' &= \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(T_{e_i}\times N_d\right)\right)_j - \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(T_{e_i}\times N_d{}'\right)\right)_j \\ &= \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(\left(T_{e_i}\times N_d\right)-\left(T_{e_i}\times N_d{}'\right)\right)\right)_j \\ &= \sum_{j=1}^{G}\left(\sum_{i=1}^{N_e}\left(T_{e_i}\times\left(N_d - N_d{}'\right)\right)\right)_j > 0. \end{aligned} \tag{11}$$

3) Guiding Principle 3: To achieve a fair selection of the chromosomes, an input data set must be segmented into equally sized chunks.

a) Supporting argument: The evolutionary process progresses with an increasing number of chromosomes matching the schema H in each generation.

Let

1) F be the fitness function representing the performance of a chromosome in a generation;
2) d_i be the size of the ith data chunk;
3) c be a chromosome;
4) n be the number of chromosomes matching a schema H;
5) G is the number of chromosomes in the current generation.

Considering (5), it can be observed that $P_R(H,t)$ depends on both $\bar{f}(H,t)$ and $\bar{f}(t)$ which can be computed by

$$\bar{f}(H,t) = \frac{\sum_{i=0}^{n} F(c,d_i)}{n} \tag{12}$$

$$\bar{f}(t) = \frac{\sum_{i=0}^{G} F(c,d_i)}{G}. \tag{13}$$

As a result, the probability that a chromosome is selected for evolution depends on the size of the data chunk that the chromosome processes. To ensure a fair selection, each chromosome is processed with data chunks of the same size which leads to an efficient evolution.

IV. Data Engineering in GEP

Based on the proposed three guiding principles in Section III, we present a data engineered GEP to speed up computation in evolution.

A. Segmentation

Segmentation is employed to segment the original input data set into a number of smaller data chunks of an equal size. The size of a data chunk is determined by a predefined segmentation ratio. A data chunk consists of a number of data samples. Two segmentation approaches are employed, which are *random selection* and *cutting in sequence*. Following the approach presented in [33] which provides a good sampling performance in data coverage, *random selection* is developed to select data samples from the original input data set and generate a data chunk. Each chromosome in a generation is processed with the same data chunk during the evolution of GEP. *Cutting in sequence* is implemented to cut the original input data set into a number of data chunks of an equal size in sequence. The order of the data samples in the data chunks remains the same as they appear in the original data set. While *random selection* targets at data samples without a strong correlation, the *cutting in sequence* segmentation scheme considers the correlations among the data samples of a data chunk.

B. Overlapping

While segmentation reduces the computation complexity of GEP, processing individual data chunks instead of the whole data set normally degrades the accuracy level of GEP [1]. This is especially true when the data samples have strong correlations. To minimize the accuracy degradation of GEP in data segmentation, an overlapping scheme is developed which

Algorithm 1 Overlapping Implementation

Input: two data chunks (A, B) without overlapping;
Output: two overlapped data chunks (A, B);
1: Set an overlapping ratio;
2: Calculate the number of samples to be overlapped;
3: **FOR** x = 1 **TO** number of samples **DO**
4: Take a sample from the overlapped partition in data chunk A;
5: Overwrite the sample in the overlapped partition of data chunk B;
6: x + +;
7: **ENDFOR**
8: **RETURN** data chunks A and B;

Algorithm 2 GEP Implementation

Input: A data set;
Output: A mathematical expression;
1: Segment the input data set into N data chunks
2: Generate N overlapped data chunks
3: Initialize the first generation of the population with more than N chromosomes;
4: best_chromosome = chromosome(1);
5: best_fitness_value = 0;
6: **WHILE** i< termination generation number **DO**
7: **FOR** x = 1 **TO** size of the current population **DO**
8: Translate chromosome(x) into an expression tree(x);
9: global_fitness_value(x) = fitness_evaluator(expression_tree(x), N data_chunks);
10: **IF** global_fitness_value(x)=the number of samples in data_chunk(x) **THEN**
11: best_chromosome = chromosome(x) **GOTO 21**;
12: **ELSE IF** global_fitness_value(x) > best_fitness_value **THEN**
13: best_chromosome = chromosome(x);
14: best_fitness_value = global_fitness_value(x);
15: **ENDIF**
16: x + +;
17: **ENDFOR**
18: Generate the population of the next generation;
19: i + +;
20: **ENDWHILE**
21: **RETURN** best_chromosome;

takes into account the correlations among the data samples. Algorithm 1 presents the overlapping scheme implemented in the data engineered GEP.

C. GEP Implementation

Considering segmentation and overlapping, the data engineered GEP is implemented as shown in Algorithm 2. The GEP takes an input data set, and generates a mathematical expression which represents the correlations of the input data parameters. The fitness evaluator of line 9 assesses the performance of each chromosome in a generation following the classical fitness function proposed in [1]. This fitness evaluator has two versions, one is designed for the *random selection* segmentation scheme without overlapping, whereas the other is designed for the *cutting in sequence* segmentation scheme with overlapping. In the case of *random selection*, the quality of a chromosome is assessed considering the best local fitness value.

However, the assessment in the case of *cutting in sequence* follows the way as shown in Algorithm 3. In this case, the quality of a chromosome is assessed based on its global fitness value, which is an average of the local fitness values of the chromosome when processing all the data chunks as shown in lines 6–12. This helps prevent the GEP from trapping in a local optimum.

Algorithm 3 Fitness Evaluator

Input: N data chunks and an expression_tree(x);
Output: The fitness value of a given chromosome;
1: data_chunk_no = x mod N;
2: current_data_chunk = data_chunk(data_chunk_no);
3: local_fitness_value = fitness(expression(x), current_data_chunk);
4: fitness_value = local_fitness_value;
5: **IF** local_fitness_value > best_fitness_value **THEN**
6: **FOR** y = 1 **TO** the number of N **DO**
7: current_data_chunk = data_chunk(y);
8: local_fitness_value = fitness(expression(x), current_data_chunk);
9: accumulation = accumulation + local_fitness_value;
10: **ENDFOR**
11: average_fitness_value = accumulation / N;
12: fitness_value = average_fitness_value;
13: **ENDIF**
14: **RETURN** fitness_value;

Algorithm 4 GEP Parallelization

Input: m CPU-threads, a population of chromosomes, N data chunks;
Output: the fitness values of chromosome in a population;
1: remain_chromosome = size of the current population;
2: **WHILE** remain_chromosome>0 **DO**
3: **FOR** y = 1 **TO** the number of m **DO**
4: index = remain_chromosome;
5: Assign CPU-Thread(y, chromosome (index)) //parallel execution
6: {
7: Translate chromosome(index) into an expression tree(index);
8: fitnese_value=fitness_evaluator(expression_tree(index), N data_chunks);
9: global_fitnese_value = fitness_value;
10: }
11: remain_chromosome = remain_chromosome − 1;
12: **ENDFOR**
13: **ENDWHILE**
14: **RETURN** global_fitnese_value;

D. GEP Parallelization

The data engineered GEP presented in Section IV-C is further parallelized with an aim to speed up the computation process when dealing with potential big data. The parallel GEP maintains the generation structure in such a way that it processes the chromosomes on a generation basis using a number of CPU cores simultaneously of which each CPU core has two threads. The multithreaded OpenMP [36] is employed in the parallelization of the GEP calculating the fitness values of the chromosomes of a generation in parallel as shown in Algorithm 4.

V. Performance Evaluation

To evaluate the performance of the data engineered GEP, a number of experiments were conducted. This section analyzes the impact of segmentation, overlapping, and parallelization on the performance of the GEP, respectively. First, it introduces the two data sets employed in the evaluation.

A. Data Sets

Two data sets were evaluated in the experimental tests which are detailed below.

1) Power System Data Set: The total data set contains 9568 data points (measurements) collected from a combined cycle power plant over six years [34], [35]. It consists of 5000 measurements for training and 4568 measurements for testing. Following our previous work presented in [39],

TABLE I
GEP PARAMETER SETTINGS

Parameters	Values	
Population size	100	
No. of genes in a chromosome	1	
No. of generations	Physics data	20000
	Power system data	10000
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%

GEP generates a mathematical function which represents the correlations of the power-related environmental factors for production prediction of the power plant.

2) Particle Physics Data Set: This data set [17]–[19] contains 10 000 samples of events of which the first 5000 samples were used for training and the rest were used for testing. A sample can be classified into an event signal or a background noise. Each sample has eight input factors. Similar to the processing on the power system data set, the data engineered GEP also generates a mathematical function representing the correlations of the input factors which is used for classification.

It is worth noting that the use of two data sets in the evaluation has some considerations. On one hand, the time serial power system data set is not complex in that each data sample has a small number of factors with simple mathematical dependencies. However, the power data samples have a strong correlation among them. On the other hand, the particle physics data set is complex due to the large size of input factors of a data sample together with the mathematical or logical dependencies among these factors. Different from the power data set, the samples in the particle physics data set are not highly correlated. As a result, these two data sets with complementary features were selected for evaluating the performance of the data engineered GEP.

B. GEP Parameter Settings

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

One gene was employed for each chromosome to avoid the use of the connection function, which might lead to an inefficient chromosome structure [1]. Considering the complexity of the two data sets, we set 20 000 generations for the physics data and 10 000 generations for the power data.

To evaluate the performance of the data engineered GEP, an Intel Xenon Server was configured with two Intel E5-2697 V2 CPU processors at 2.7 GHz running Linux Ubuntu version 14.04. Each of the two processors has 12 CPU cores and supports 24 threads with a shared memory space of 64 GB. We conducted ten runs for each test in the evaluation and observed that the execution times of the ten runs were highly stable. For example, Table II shows the coefficient of variation values of nine tests on the two data sets, which are in the range between 2.2% and 10.8%. As a result, an average value of ten runs was taken for each test.

TABLE II
COEFFICIENT OF VARIATION VALUES (%)

Number of CPU threads	1	2	3	4	8	12	16	24	48
Particle physics data	4.4	3.7	3.7	2.8	8.6	5.4	4.6	2.2	5.4
Power system data	10.8	10.5	8.3	9.6	8.8	7.8	6.2	6.9	8.3

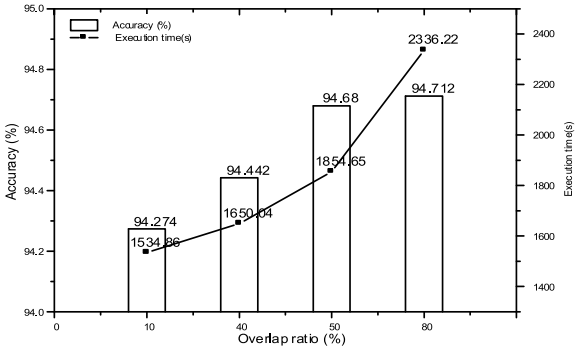


Fig. 2. Impact of overlapping on particle physics data.

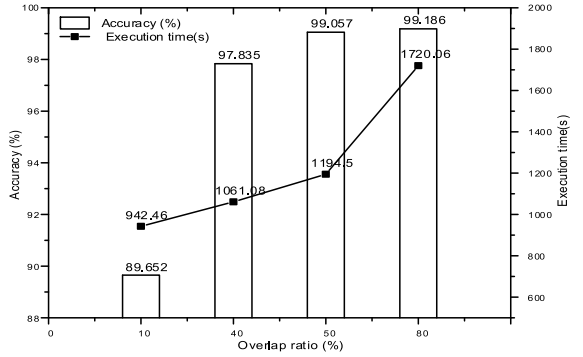


Fig. 3. Impact of overlapping on power system data.

C. Overlapping

A number of tests were conducted to evaluate the performance of the GEP with the *cutting in sequence* overlapping scheme from the aspects of both accuracy and execution time. Figs. 2 and 3 show the results of the GEP on the two data sets with a segmentation ratio of 10%.

From Figs. 2 and 3, it can be observed that accuracy level of the GEP goes up with an increasing overlapping ratio on the two data sets but at the cost of a higher execution time in computation. The overlapping ratios of 10%, 40%, 50%, and 80% were evaluated with a consideration that a low or high overlapping ratio would not balance well the tradeoff between the accuracy gain and execution time incurred. That was the reason why 50% was selected as the best overlapping ratio.

D. Segmentation

The segmentation ratio determines the size of a data chunk that is assigned to each chromosome. Three segmentation

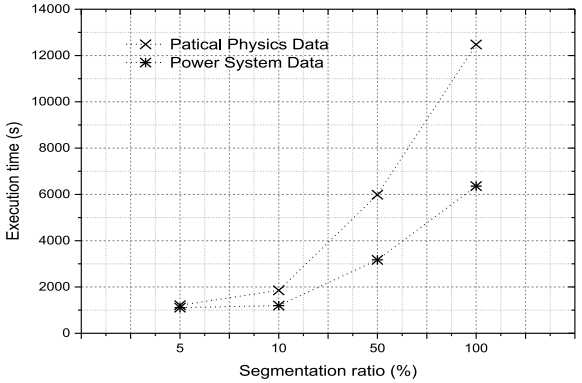


Fig. 4. Impact of segmentation on the two data sets.

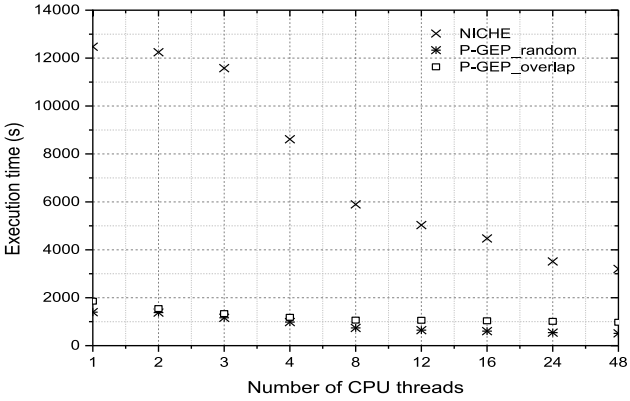


Fig. 5. Computation of the P-GEP on particle physics data.

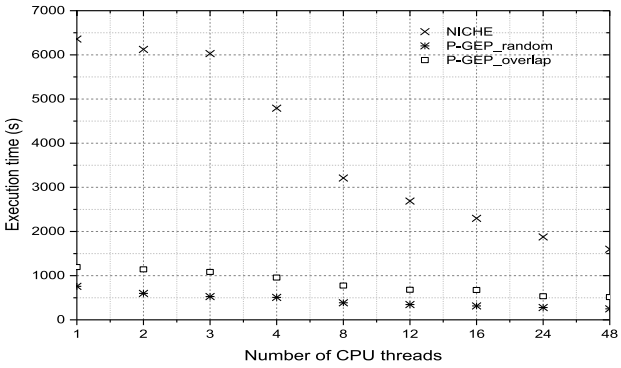


Fig. 6. Computation of the P-GEP on power system data.

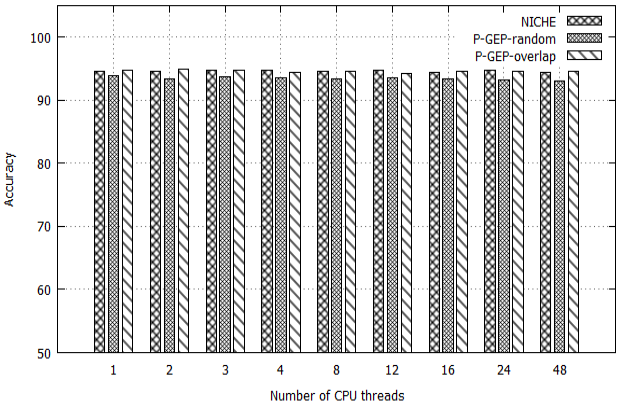


Fig. 7. Accuracy of the P-GEP on particle physics data.

ratios (i.e., 50%, 10%, and 5%) were tested in the evaluation. Fig. 4 shows the impacts of the segmentation ratios on the execution time of the data engineered GEP on the two data sets, respectively.

Although the execution time of GEP decreases when the segmentation ratio goes down, a small segmentation ratio might lead to a low accuracy level in data processing. For example, when the segmentation ratio is 10%, the GEP produces an accuracy of 94.68% on the particle physics data and 99.06% on the power system data, respectively. However, the case of using a segmentation ratio of 5% generates 93.942% on the particle physics data and 96.81% on the power system data in term of accuracy. As a result, a segmentation ratio of 10% was selected in the evaluation.

E. Parallelization

To evaluate the performance of the data engineered GEP in parallelization (denoted as P-GEP), we implemented an existing parallel GEP work (i.e., NICHE) [27] for comparison purpose. The number of CPU threads was varied from 1 to 48 in the tests. Two versions of the P-GEP were implemented. The P-GEP-*overlap* adopts the *cutting in sequence* segmentation scheme with overlapping, whereas the P-GEP-*random* adopts the *random* segmentation scheme without overlapping.

It can be observed from Figs. 5 and 6 that the execution time of the P-GEP in processing both the particle physics data and the power system data decreases with an increasing number of CPU threads. The two versions of the P-GEP are significantly faster than the NICHE work. This is mainly due to the fact that P-GEP follows closely the generation structure of GEP leading to an efficient evolution. In addition, processing segmented data chunks further speeds up the computation. P-GEP-*random* is even faster than P-GEP-*overlap* because the less computation overhead incurred in accessing the multiple data chunks. It is worth noting that the execution time of the P-GEP in processing small data chunks does not decrease significantly when the number of CPU threads increases which reflects the fact that parallelization better suits big data processing which will be further discussed in Section VI.

Figs. 7 and 8 show the accuracy of P-GEP in comparison with the NICHE work in processing the two data sets. The accuracy of P-GEP-*overlap* is similar to that of NICHE in all the tests. On average, P-GEP-*overlap* produces an accuracy of 94.57% on the particle physics data and 96.26% on the power system data, whereas NICHE produces an accuracy of 94.62% and 94.83%, respectively. It is worth noting that P-GEP-*overlap* is more accurate than P-GEP-*random* on the power system data due to the fact that overlapping well suits data sets, such as the power system data with a strong correlation among data samples. The P-GEP-*random* produces the worst level of accuracy due to its random selection of data chunks without overlapping.

Figs. 9 and 10 further show that parallelization better suits for processing potential big data. It can be observed from Fig. 9 that the execution time of P-GEP-*overlap* using

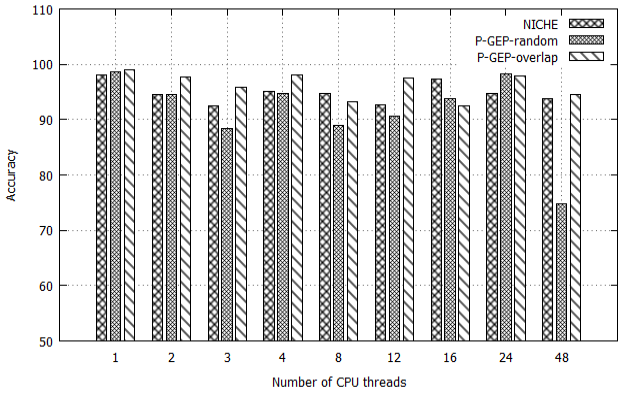


Fig. 8. Accuracy of the P-GEP on power system data.

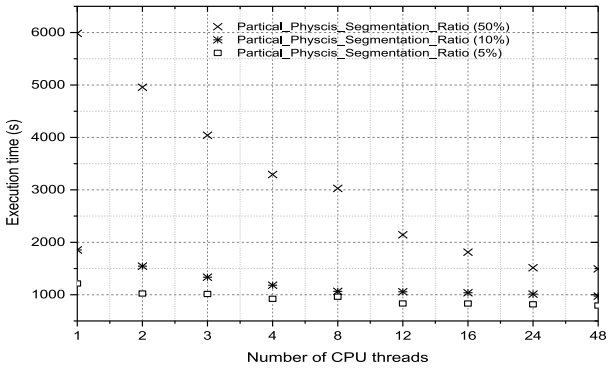


Fig. 9. Impact of segmentation ratio on the execution time of P-GEP-*overlap* in processing particle physics data.

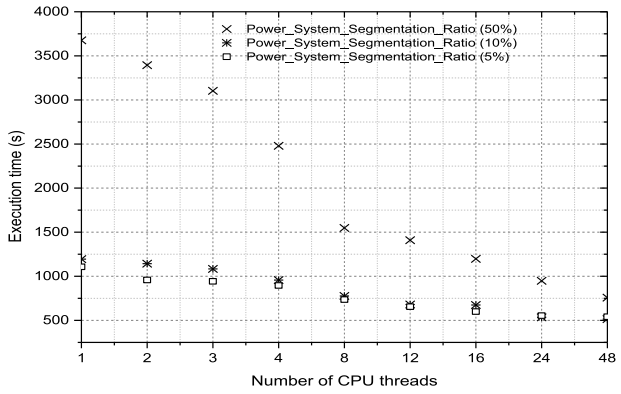


Fig. 10. Impact of segmentation ratio on the execution time of the P-GEP-*overlap* in processing power system data.

a segmentation ratio of 50% decreases significantly when the number of CPU threads increases. However, P-GEP-*overlap* does not produce much difference in processing the particle physics data using a segmentation of 10% and 5%, respectively. In the case of processing power system data as shown in Fig. 10, the execution time of the parallel P-GEP-*overlap* using a segmentation ratio of 5% is even slower than the case of using a segmentation ratio of 10% when the numbers of CPU threads are 24 and 48, respectively. This is because the segmented power system data chunks are small in volume which leads to a higher overhead in parallelization than the speedup achieved in computation.

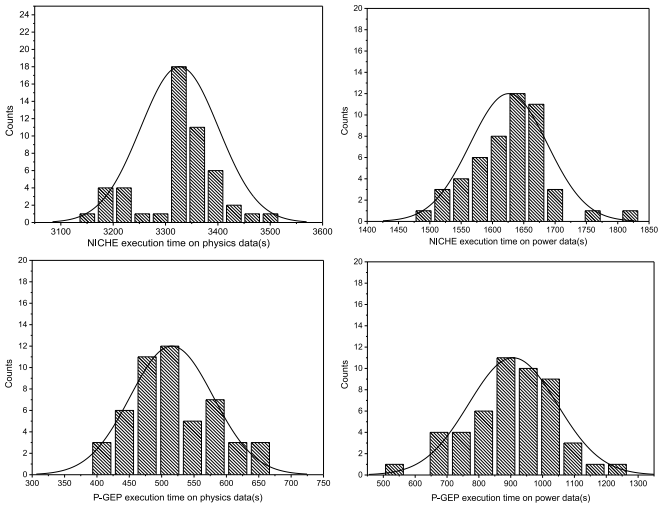


Fig. 11. Distributions of the execution times of the P-GEP and NICHE in processing the two data sets.

TABLE III
RESULTS OF SHAPIRO–WILK TEST

Samples	*W* values
NICHE (physics data)	0.920
NICHE (power data)	0.958
P-GEP (physics data)	0.988
P-GEP (power data)	0.957

TABLE IV
RESULTS OF *t*-TEST

	Execution Time (s)			Accuracy (%)	
	mean	*t*-value	significance level (%)	mean	standard deviation
NICHE (physics data)	3326.74	108.5	99.99	94.49	0.622
P-GEP (physics data)	906.69			94.51	0.670
NICHE (power data)	1625.99	4.506	99.99	96.65	8.685
P-GEP (power data)	515.47			96.81	3.860

F. Statistical Analysis

To further compare the performance of the data engineered GEP with that of the NICHE work, we employed 48 CPU threads and conducted 50 runs in total on the two data sets, respectively. The execution times in running the two algorithms follow a normal distribution as can be observed from Fig. 11.

We further performed normality test on the execution times of the two algorithms using the Shaprio–Wilk test [46] which handles well with a small number of data samples. The *W* values of the Shaprio–Wilk tests as shown in Table III confirm the observed normal distributions as shown in Fig. 11.

Therefore, we employed *t*-test [45] to compare P-GEP with NICHE on the execution times which follow a normal distribution and the comparison results are shown in Table IV. It can be observed that the data engineered GEP with overlapping is faster than NICHE on both data sets at a significance level higher than 99.9%. We further observe that the accuracy of the data engineered GEP is slightly higher and more stable than that of NICHE. This is mainly due to the fact

that the data engineered GEP considers the global fitness of chromosomes rather than their local values.

VI. GEP Computation Scalability Analysis

To further investigate the computation scalability of the data engineered GEP in dealing with potential big data using a large number of CPU threads, we developed its computation model based on the experimental results presented in Section V. In this section, we present the computation model and analyze the computation scalability of the data engineered GEP.

A. GEP Computation Model

Following our previous work [39] we developed a computation model of the data engineered GEP on the two data sets, respectively, which represents the correlations between the input parameters (number of CPU threads x_0, data size x_1, and segmentation ratio x_2) and the output (execution time).

The computation model of the data engineered GEP for the particle physics data set can be represented by

$$\begin{aligned}\text{ExecutionTime} = &\left[\frac{-49.6019773651}{\text{Sin}(x_0)} * \text{Sin}(x_1)\right] \\ &+ 2 * \left(\frac{\text{Square}(x_0)}{\text{Sqrt}(x1)}\right) \\ &+ \text{Square}(25.013766624) \\ &+ \text{Sqrt}\left(\frac{\text{Power}(20.9463112056, 4)}{\text{Sqrt}(x_1)}\right). \quad (14)\end{aligned}$$

This is mined from the experimental results obtained using both 5% and 50% segmentation ratios on the physics data. These two ratios generated a large gap between the two result sets which leads to a highly accurate computation model in dealing with data samples with a large number of factors.

For the power system data, we employed the experimental results obtained using both 5% and 10% segmentation ratios to mine the computation model of data engineered GEP which can be represented by

$$\begin{aligned}\text{ExecutionTime} = &\text{Power}(\text{Sqrt}(x_1 - \text{TAN}(x_2)), 3) \\ &+ \left(\left(\frac{x_0 - x_1}{\text{Sqrt}(x_0)}\right) * (x_0 - x_2)\right) \\ &+ \text{Cos}(\text{Log}(x_1)) * (\text{Square}(x_0)) \\ &+ \text{Cos}(\text{Power}(x_1, -401043.774094)) * \left[\text{Square}(x_0)\right] \\ &- \left(\text{Tan}\left(\text{Log}\left[\text{Square}\left(\frac{80595.3126401}{x_1}\right)\right]\right)\right) \\ &+ \text{Square}\left(\text{Log}\left(\text{Square}\left[\frac{-409114.183858}{x_1}\right]\right)\right) \\ &+ \text{Square}\left(\text{Log}\left(\text{Square}\left[\frac{-22415.3897725}{x_1}\right]\right)\right)\Bigg) \\ &\times \frac{x_2}{100} * \frac{x_0}{80.6} * \left(\text{Power}\left(1 + 0.6 * \frac{1}{x_0}, x_0 - 1\right)\right. \\ &\left. - \text{Power}(1.0093, x_0)\right). \quad (15)\end{aligned}$$

The use of these two ratios on the power system data with a small gap aimed to reflect the fine-grained behaviors of the computation model in dealing with data samples with a small number of factors.

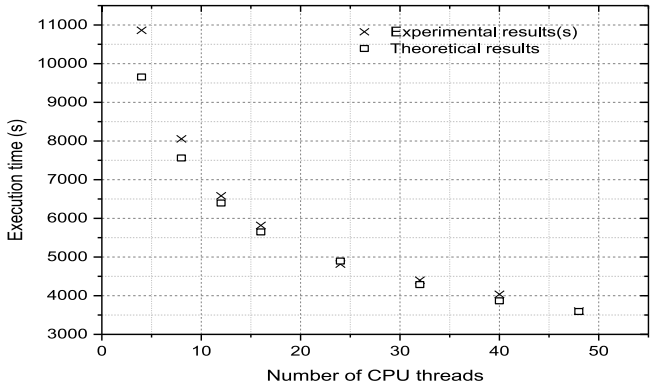


Fig. 12. Computation model validation on particle physics data.

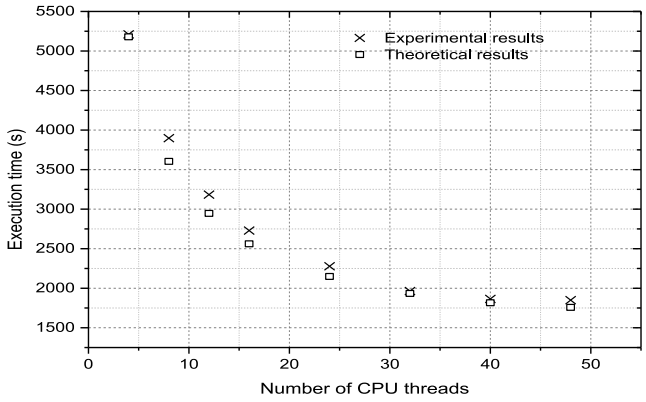


Fig. 13. Computation model validation on power system data.

B. Validation of GEP Computation Model

We employed the two data sets of the original sizes to generate the computation model to estimate the execution times of the data engineered GEP running on a varied number of CPU threads. To validate the computation model of the data engineered GEP, we compared the estimated values with the actual execution times in processing the two data sets but with doubled sizes. Figs. 12 and 13 show the performance of the computation model on the two data sets, respectively, using a segmentation ratio of 50%.

The accuracy of the computation model can be computed by

$$\begin{aligned}\text{Accuracy} = &\ 100\% \\ &- \left(\frac{|\text{Theoretical Result} - \text{Experimental Result}|}{\text{Experimental Result}}\right) \\ &\times 100\%. \quad (16)\end{aligned}$$

Tables V and VI show that the computation model achieves an average accuracy level of 96.05% on the particle physics data and 95.14% on the power system data, respectively.

C. Computation Scalability

We applied the computation model to evaluate the scalability of the data engineered GEP in dealing with big data scenarios. Figs. 14 and 15 show that for the two data sets,

TABLE V
COMPUTATION MODEL VALIDATION ON PARTICLE PHYSICS DATA

Number of threads	4	8	12	16	24	32	40	48
Accuracy level (%)	88.88	93.87	97.17	97.12	98.41	97.46	95.80	99.66
Average (%)	96.05							

TABLE VI
COMPUTATION MODEL VALIDATION ON POWER SYSTEM DATA

Number of threads	4	8	12	16	24	32	40	48
Accuracy level (%)	99.40	91.84	91.93	93.44	94.02	98.42	94.31	94.78
Average (%)	95.14							

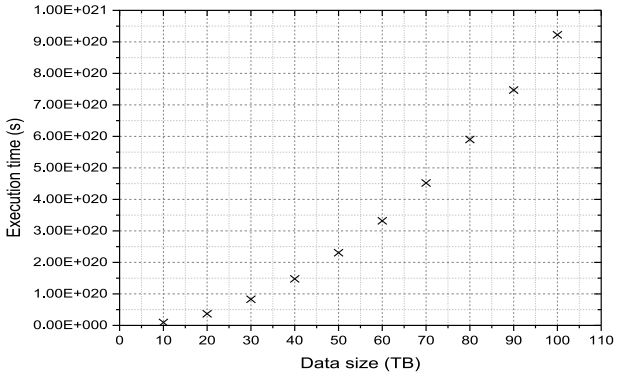


Fig. 14. Computation scalability on particle physics data.

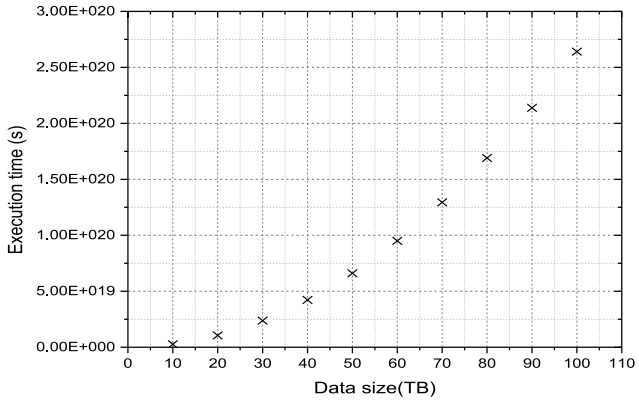


Fig. 15. Computation scalability on power system data.

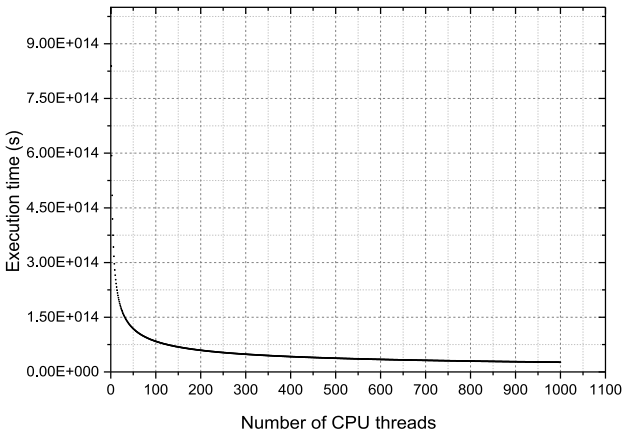


Fig. 16. Parallelization on particle physics data.

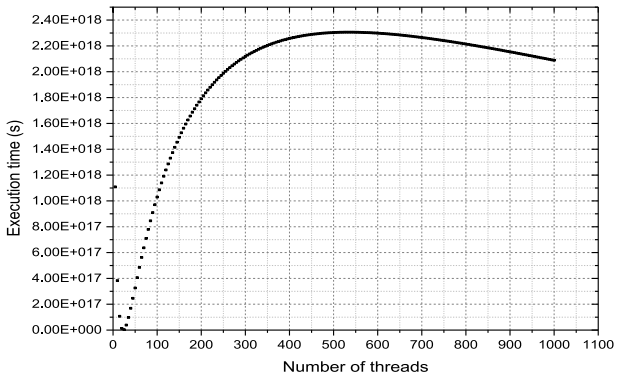


Fig. 17. Parallelization on power system data.

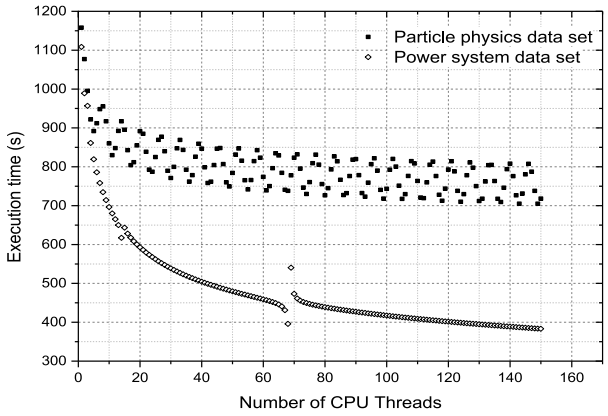


Fig. 18. Fluctuations in performance gain via parallelization.

the execution time of the data engineered GEP increases slowly with an increasing size of input data up to 100 TB, using 10 000 CPU threads.

We further evaluated the computation scalability of the data engineered GEP in dealing with varied numbers of CPU threads. Fig. 16 shows that the execution time of the data engineered GEP decrease when processing 1 TB particle physics data with an increasing number of CPU threads up to 1000. It can be observed that the speedup of parallelization is high when the number of CPUs is less than 100 due to the fact that CPU threads themselves can also cause an additional computation overhead.

Data samples in the power system data set have a simpler structure than the data samples in the particle physics data set. As a result, the performance gain achieved via parallelization in processing one unit of power system data using a number of CPU threads is less than the case of processing one unit of particle physics data. When the structure of a data set like the power system data is simple, the performance gain of parallelization can be easily offset by the computation overhead incurred in maintaining these CPU threads. This can be observed from Fig. 17 showing that the execution time of the data engineered GEP decreases sharply with an increasing

number of CPU threads up to 23. The data engineered GEP reaches the lowest estimated execution time of 5.66E + 013 s when 23 CPU threads participate in the computation. After this point, the execution time goes up due to a high ratio of the overhead incurred in maintaining these CPU threads to the performance gain achieved through parallelization. The fluctuations in performance gain via parallelization can be further observed in Fig. 18, where a segmentation ratio of 5% was used on the two original data sets.

Overall the data engineered GEP achieves a high scalability in dealing with potential big data using a large number of CPU threads.

VII. Conclusion

In this paper, we have presented an efficient data engineered GEP solution in dealing with potential big data. It builds on the proposed three guiding principles which necessitate the considerations on the generation structure of chromosomes, the size of input data, and the segmentation of data chunks when speeding up the evolution process of GEP. Experimental results confirmed that the data engineered GEP which follows closely the generation structure of chromosomes in evolution and considers the size of input data did speed up the evolution process significantly without loss of accuracy in data correlation mining. The computation model further showed that the data engineered GEP is highly scalable in dealing with potential big data.

It should be pointed out that for data sets with a high volume in size but a low complexity in data structure, purely increasing the number of CPU threads could lead to slow executions due to the fact that the overhead incurred in maintaining these CPU threads is higher than the performance gain to be achieved through parallelization.

The data engineered GEP can further benefit from the schema theory proposed in our previous work [23] which introduces the concept of building blocks in GEP evolution. A GEP building block is a segment shared by high quality chromosomes in a population which can be discovered during the evolutionary process. Building blocks can be used to replace the corresponding segments of low quality chromosomes for computation speedup in evolution. Therefore, a future work will research how the data engineered GEP can be integrated with building blocks.

References

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

[2] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, U.K.: Oxford Univ. Press, 1996.

[3] 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.

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

[5] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

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

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

[8] 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.*, Las Vegas, NV, USA, Jun. 2002, pp. 1355–1361.

[9] 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.

[10] 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. (AIML)*, Cairo, Egypt, 2005, pp. 11–16.

[11] J. Zuo, C.-J. Tang, C. Li, C.-A. Yuan, and A.-L. Chen, "Time series prediction based on gene expression programming," in *Proc. 5th Int. Conf. Adv. Web Age Inf. Manag. (WAIM)*, vol. 3129. Dalian, China, 2004, pp. 55–64.

[12] 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 Acquisition Adv. Comput. Syst. Technol. Appl.*, Sofia, Bulgaria, 2005, pp. 133–138.

[13] H. S. Lopes and W. R. Weinert, "A gene expression programming system for time series modeling," in *Proc. 25th Iberian Latin Amer. Congr. Comput. Methods Eng. (CILAMCE),* Recife, Brazil, 2004, pp. 1–13.

[14] 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.

[15] Z. Cai, Q. Li, S. Jiang, and L. Zhu, "Symbolic regression based on GEP and its application in predicting amount of gas emitted from coal face," in *Proc. Int. Symp. Safety Sci. Technol.*, 2004, pp. 637–641.

[16] 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. (SYNASC)*, Timişoara, Romania, 2005, pp. 321–324.

[17] L. Teodorescu and Z. Huang, "Enhanced gene expression programming for signal-background discrimination in particle physics," in *Proc. 12th Adv. Comput. Anal. Tech. Phys. Res.*, 2008, pp. 1–11.

[18] 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.

[19] L. Teodorescu, "High energy physics data analysis with gene expression programming," in *Proc. IEEE Nucl. Sci. Symp. Conf. Rec.*, vol. 1. Fajardo, Puerto Rico, 2005, pp. 143–147.

[20] X. Li, C. Zhou, W. Xiao, and P. C. Nelson, "Prefix gene expression programming," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Washington, DC, USA, 2005, pp. 25–31, 2005.

[21] L. Huo, J. Yin, L. Guo, J. Hu, and X. Fan, "Short-term load forecasting based on improved gene expression programming," in *Proc. IEEE Int. Conf. Circuits Syst. Commun.*, Shanghai, China, 2008, pp. 5647–5650.

[22] S. F. Mekhamer, A. Y. Abdelaziz, H. M. Khattab, and M. A. L. Badr, "Gene expression programming for power system static security assessment," *Int. J. Eng. Sci. Technol.*, vol. 4, no. 2, pp. 77–88, 2012.

[23] Z. Huang, "Schema theory for gene expression programming," Ph.D. dissertation, Department of Electronic and Computer Engineering, Brunel Univ. at London, London, U.K., 2014.

[24] C. Zhihua, J. Siwei, Z. Li, and G. Yuanyuan, "A novel algorithm of gene expression programming based on simulated annealing," in *Proc. Int. Symp. Intell. Comput. Appl.*, 2005, pp. 605–610.

[25] M. Snir, S. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, vol. 2. Cambridge, MA, USA: MIT Press, 1996.

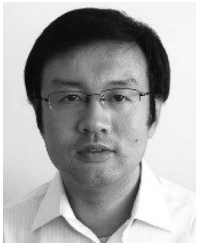
[26] X. Du, L. Ding, and L. Jia, "Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm," in *Proc. 4th Int. Conf. Nat. Comput.*, Jinan, China, 2008, pp. 433–437.

[27] J. Wu *et al.*, "Parallel NICHE gene expression programming based on general multi-core processor," in *Proc. Int. Conf. Artif. Intell. Comput. Intell. (AICI)*, vol. 3. Sanya, China, 2010, pp. 75–79.

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

[29] H. Cheng and J. Xue, "The research on evolution schema theorem on gene expression programming," in *Emerging Computation and Information Technologies for Education*, E. Mao, L. Xu, and W. Tian, Eds. Heidelberg, Germany: Springer, 2012, pp. 399–406.

[30] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," *Evol. Comput.*, vol. 6, no. 3, pp. 231–252, 1998.
[31] R. Poli and N. F. McPhee, "General schema theory for genetic programming with subtree-swapping crossover: Part I," *Evol. Comput.*, vol. 11, no. 1, pp. 53–66, Mar. 2003.
[32] R. Poli and N. F. McPhee, "General Schema theory for genetic programming with subtree-swapping crossover: Part II," *Evol. Comput.*, vol. 11, no. 2, pp. 169–206, Jun. 2003.
[33] J. S. Vitter, "An efficient algorithm for sequential random sampling," *ACM Trans. Math. Softw.*, vol. 13, no. 1, pp. 58–67, 1987.
[34] P. Tüfekci, "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *Int. J. Elect. Power Energy Syst.*, vol. 60, pp. 126–140, Sep. 2014.
[35] 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. (ICETCEE)*, 2012, pp. 13–18.
[36] D. Novillo, "OpenMP and automatic parallelization in GCC," in *Proc. GCC Developers*, 2006, pp. 1–10.
[37] R. Chandra *et al.*, *Parallel Programming in OpenMP*, vol. 129. San Francisco, CA, USA: Morgan Kaufmann, 2001.
[38] J. Zhong, Y.-S. Ong, and W. Cai, "Self-learning gene expression programming," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 65–80, Feb. 2016.
[39] 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, doi: 10.1002/cpe.3786.
[40] *Apache Hadoop*. Accessed: Jun. 28, 2016. [Online]. Available: https://hadoop.apache.org/
[41] S. W. Wilson, *Classifier Conditions Using Gene Expression Programming* (LNCS 4998). Heidelberg, Germany: Springer, 2008, pp. 206–217.
[42] J. S. Manognya and L. P. Wang, "Gene expression programming for induction of finite transducer," in *Proc. 7th Int. Conf. Inf. Commun. Signal Process. (ICICS)*, Macau, China, 2009, pp. 1–5.
[43] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 217–228, Feb. 2015.
[44] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 309–325, Jun. 2015.
[45] J. F. Box, "Guinness, gosset, fisher, and small samples," *Stat. Sci.*, vol. 2, no. 1, pp. 45–52, 1987.
[46] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, nos. 3–4, pp. 591–611, 1965.



Zhengwen Huang received the B.Sc. degree from the University of Science and Technology, Hefei, China, the M.Sc. degree from 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 2014.

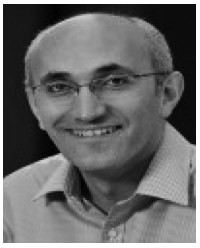
His current research interests include evolutionary algorithms (gene expression programming and genetic programming) and data engineering.

Maozhen Li received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 1997.

He is currently a Professor with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K. He was a Post-Doctoral Research Fellow with the School of Computer Science and Informatics, Cardiff University, Cardiff, U.K., from 1999 to 2002. His current research interests include high performance computing, big data analytics, and intelligent systems. He has over 150 research publications in the above areas.

Dr. Li is on the editorial boards of a number of journals. He is a fellow of the British Computer Society.

Christos Chousidis received the Ph.D. degree from the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K., in 2013. He is currently a Senior Lecturer in Applied Sound Engineering in the School of Computing and Engineering at the University of West London, U.K. His research is on data engineering with a special focus on audio wireless networks. He is a member of the Audio Engineering Society.



Alireza Mousavi received the Ph.D. degree from the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K., in 1998.

He is a Reader of Systems Engineering and Computing. His current 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.

Dr. Mousavi is a member of the Institute of Engineering and Technology (IET).

Changjun Jiang received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995.

He was a Post-Doctoral Research Fellow with the Institute of Computing Technology, Chinese Academy of Sciences in 1997. He is currently a Professor with the Department of Computer Science and Engineering, Tongji University, Shanghai, China. He is also a Council Member of the China Automation Federation and Artificial Intelligence Federation, the Director of the Professional Committee of Petri Net of the China Computer Federation, and the Vice Director of the Professional Committee of Management Systems of the China Automation Federation. He was a Visiting Professor with the Institute of Computing Technology, Chinese Academy of Science, a Research Fellow with the City University of Hong Kong, Hong Kong, and an Information Area Specialist of the Shanghai Municipal Government. His current research interests include concurrent theory, Petri net, and formal verification of software, concurrency processing, and intelligent transportation systems.

Dr. Jiang is a fellow of the IET.