Chapter 9
Genetic Programming Based on Error Decomposition: A Big Data Approach

Amirhessam Tahmassebi and Amir H. Gandomi

Abstract An investigation of the deviations of error and correlation for different stages of the multi-stage genetic programming (MSGP) algorithm in multivariate nonlinear problems is presented. The MSGP algorithm consists of two main stages: (1) incorporating the individual effect of the predictor variables, (2) incorporating the interactions among the predictor variables. The MSGP algorithm formulates these two terms in an efficient procedure to optimize the error among the predicted and the actual values. In addition to this, the proposed pipeline of the MSGP algorithm is implemented with a combination of parallel processing algorithms to run multiple jobs at the same time. To demonstrate the capabilities of the MSGP, its performance is compared with standard GP in modeling a regression problem. The results illustrate that the MSGP algorithm outperforms standard GP in terms of accuracy, efficiency, and computational cost.

9.1 Introduction

We are entering the era of big data that refers to the explosion of available information with new promising levels of scientific exploration. Despite the novel opportunities that big data offers to recent society, it brings challenges including computational cost, huge high-dimensional sample size, storage impasse, and error extent. The rise of big data in various scientific fields such as genomics, economics, finance, neuroscience, internet security, digital humanities, etc and their challenges

A. Tahmassebi
Department of Scientific Computing, Florida State University, Tallahassee, FL, USA
e-mail: atahmassebi@fsu.edu

A. H. Gandomi (✉)
School of Business, Stevens Institute of Technology, Hoboken, NJ, USA

BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, MI, USA
e-mail: a.h.gandomi@stevens.edu

W. Banzhaf et al. (eds.), *Genetic Programming Theory and Practice XV*,
Genetic and Evolutionary Computation, https://doi.org/10.1007/978-3-319-90512-9_9

demand new evolutionary computational paradigms to deal with salient features of big data, including heterogeneity, noise accumulation, spurious correlation, and incidental endogeneity [3]. Evolutionary algorithms have mostly been successful in solving big data problems [9, 16, 21].

Genetic programming (GP) [12] is as an extension of genetic algorithms (GA) which uses computer programs to solve problems. GP uses tree structures to represent the solutions and evolves them during generations. Prediction of real values based on each tree is the procedure by which GP performs regression [4–6, 18]. In 1998, Ryan et al. [15] proposed a relatively novel evolutionary computation method known as grammatical evolution (GE). GE provides a solution by restricting the search space using domain knowledge according to a user-specified grammar for evolving solutions. Due to the modular approach of GE, it has been successfully applied to financial applications such as predicting corporate bankruptcy, forecasting stock indices, and bond credit ratings. In addition to this, Ferreira has previously proposed another promising variant of GP, known as gene expression programming (GEP) to model nonlinear problems.

Besides the traditional tree-based GP, a linear variant of GP, know as LGP was published in Brameier and Banzhaf in 2007 [1]. The standard GP model expresses the functional programming language using tree structures in which inner nodes hold functions and leaves are the location of input predictor values. On the other hand, an evolutionary GP variant of a sequence of instructions from an imperative programming language is the essential basis for LGP. The term “linear” refers to the imperative program representation which does not mean that the method provides linear solutions [1]. Furthermore, GP has a phenomenal ability in model selection from a pool of a given population. Many of the GP-based models incorporate all the predictor input values in the modeling phase. Gandomi and Alavi [4] have previously proposed a novel scheme to formulate a problem using individual predictor variables and the interactions among them.

Different genetic operators play an essential role in the evolution process. Iba et al. [11] presented a novel method for GP, known as structured representation on genetic algorithms for nonlinear function fitting (STROGANOFF) by recombining standard GP with local hill-climbing. They have pointed out the critical changes in the semantics due to mutation operators. To overcome these difficulties: (1) they have tuned local parameters with the help of statistical identification techniques, and (2) they have controlled tree growth in GP by setting the fitness score to a minimum description length (MDL) measure. The authors validated their proposed method by comparing STROGANOFF’s effectiveness in its application to symbolic regression of nonlinear problems with numerical results. Moreover, the complexity measure can be improved by tweaking the fitness function through evolution. For example, Zhang et al. [22] analyzed fitness functions on error landscapes and the complexity measures by benchmarking the importance of tree representations of GP models via a Bayesian framework. This flexibility helped to investigate the solutions to programs which might end with bloat phenomenon. In addition to this, Zhang et al. [22] improved the fitness score by balancing the complexity of the model using an adaptive learning strategy. In this procedure, the parsimony coefficient was increased to reach better accuracy. The effectiveness of their method has been tested on real-world medical diagnosis problems.

In addition to the reasonable performance of GP models in regression problems, they have also shown great performance in classification problems in various real-world and big data problems, including those in neuroscience and medical imaging. Tahmassebi et al. [19] have employed several data reduction algorithms to reduce the dimensionality of an fMRI big data classification problem. In particular, the problem with high numbers of dimensions ($\sim$240,000) was decomposed into a new problem with feasible numbers of dimensions ($<$30) via data reduction algorithms. Then, the decomposed data were used as input predictor variables for the GP classifier. Tahmassebi et al. [20] have also shown the performance of GP in classification for large numbers of generations ($\sim$13,000) using high-performance computing (HPC). This would suggest employing parallel algorithms for such population-based evolutionary algorithms to overcome the curse of dimensionality.

In this study, we propose a GP-based scheme to decompose the error in a multivariate nonlinear problem. We apply the MSGP method, previously proposed by Gandomi and Alavi [4], in solving problems with N inputs in which N 1-dimensional programs were used instead of solving the problem with one N-dimensional program. In particular, the MSGP method incorporates the individual effect of each of the input predictor variables, and the interactions among them. Additionally, it is presented that the interactions among the input predictor variables can be neglected. This decreased the computational cost dramatically without losing more than a negligible amount of accuracy. The performance of the MSGP method is tested in a problem where the deviations of the error and correlation in each stage of the MSGP method are investigated. This opens new approaches with less computational cost and the same accuracy to tackle big data problems.

9.2 Computational Model

A multi stage evolutionary algorithm, called MSGP, is presented to decompose the error through several steps. The MSGP algorithm is implemented in Python along with GPlearn and Scikit-Learn [13] libraries. The model starts with generating a population of tree-like programs to represent the data based on stochastic formulations of variables. Just a subset of the generated programs compete with each other based on the tournament size, and the winners are optimized recursively through the evolutionary process based on the fitness metrics. Three different options were set as fitness metrics: mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). Additionally, the code has the ability of customizing the fitness metric by user-defined functions.

To find the best mathematical formulation and the fittest individual, different genetic operators such as crossover, subtree mutation, hoist mutation, and point mutation were employed in the GP model. To see the convergence during the evolutionary process, the size of the programs was increased which is normally expected

Table 9.1 Parameters setting for the GP and MSGP algorithms

Parameter	Setting
Population size	300
Number of generations	100
Tournament size	20
Crossover probability	0.7
Subtree mutation probability	0.1
Hoist mutation probability	0.05
Point mutation probability	0.1
Point replace probability	0.05
Parsimony coefficient	0.001
Stopping criteria	0.0
Max samples	0.9
Random state	1367
Number of jobs	1
Loss metric	MAE
Score metric	R^2
Function set	$+, -, \times, /$

to increase fitness values. Sometimes, this would never happen since it would cause computational costs and would make the final programs less understandable, a phenomenon called “bloat”. To control bloat, a parsimony coefficient was defined for the GP model which made programs with lower values for the fitness metric unavailable for the selection at each generation. The other alternative was generating an offspring by applying the hoist mutation operator to insert a random subtree into the original subtree location in the next generation [14]. The MSGP algorithm was implemented in Python employing parallel algorithms to run multiple jobs at the same time. By changing the number of jobs in the code, we could use the maximum CPU cores available to decrease the computational runtime which would help to solve a big data problem. The coefficient of determination (R^2) was defined as the output score for regression problems. Table 9.1 lists the parameter setting used in the MSGP model. Figure 9.1 also presents a schematic tree structure of a program. Most of the GP models discussed in Sect. 9.1 employed all the predictor variables as inputs. This incorporation of all the variables might affect the decomposition cost throughout the modeling process. To address these issues, an MSGP strategy was proposed to model the predictor variables by taking into account the effect of each of the individual predictor variables.

The MSGP algorithm could be divided into two phases:

1. Incorporating the individual effect of the predictor variables (MSGP$_{wo-int}$).
2. Incorporating the interactions among the predictor variables (MSGP$_{w-int}$).

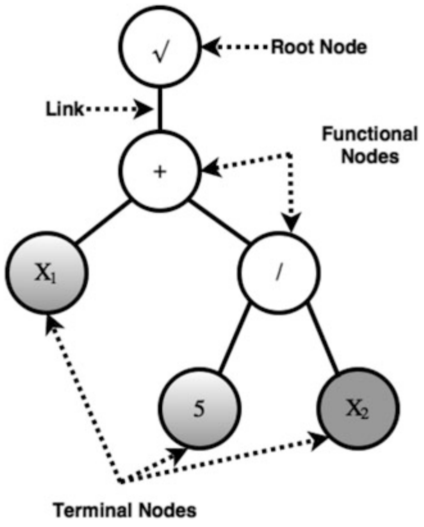

Fig. 9.1 A schematic tree representation of a GP model for $(\sqrt{X_1 + \frac{5}{X_2}})$

To shine some light on the details of the MSGP algorithm, it should be noted that the final solution, $f(X)$, entails the following terms:

$$f(X) = f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n) + f_{int}(X) = \sum_{i=1}^{n} f_i(x_i) + f_{int}(X) \tag{9.1}$$

where x_i is the input variable, n is the number of input variables, $f_i(x_i)$ indicates the function based on only one input variable x_i, and the interaction among the input variables was defined by $f_{int}(X)$. It is always possible to formulate a set of variables in terms of output values and a subset of the variables. Equation (9.2) presents the formulation of a binary problem with two variables based on the values predicted by the first input variable and the target output values:

$$f_2(x_2) = f(X) - f_1(x_1) \tag{9.2}$$

In other words, Eq. (9.2) demonstrates that a new variable formulates the error between the predicted and the actual values. This formulation is known as the decomposition of errors. This procedure can be extended by repeating the formulation presented in Eq. (9.2).

$$f_3(x_3) = f(X) - f_1(x_1) - f_2(x_2) \tag{9.3}$$

$$\vdots$$

$$f_n(x_n) = f(X) - f_1(x_1) - f_2(x_2) - \cdots - f_{n-1}(x_{n-1}) = f(X) - \sum_{i=1}^{n-1} f_i(x_1) \tag{9.4}$$

Algorithm 9.1 Multi-stage genetic programming (MSGP)

```
1  begin
2      $Y = f(X)$ ;
3      for $i = 1 : n$ (n is the number of input variables) do
           Input  : $x_i$
           Output: $Y$
4          Run GP for $f_i(X_i)$ ;
5          Generate Initial Population ;
6          Calculate Fitness of Population ;
7          if The Termination or Convergence Conditions are not satisfied then
8              Select Individuals based on Fitness;
9              Apply Genetic Operators: (Crossover, Mutation,...) ;
10             Check the Fitness of Population ;
11         end
12         $Y \leftarrow Y - f_i(x_i)$ ;
13     end
       Input  : $X\ (x_1, x_2, \ldots, x_n)$
       Output: $Y$
14     Run GP for $f_{int}(X)$ ;
15     Generate Initial Population ;
16     Calculate Fitness of Population ;
17     if The Termination or Convergence Conditions are not satisfied then
18         Select Individuals based on Fitness;
19         Apply Genetic Operators:(Crossover, Mutation,...) ;
20         Check the Fitness of Population ;
21     end
22     $f_{MSGP}(X) \leftarrow \sum_{j=1}^{n} f_j(x_j) + f_{int}(X)$ ;
23 end
```

Considering $f_{int}(X)$ presented in Eq. (9.1), the final solution calculated by the MSGP algorithm can be presented as follows:

$$f_{MSGP}(X) = \sum_{i=1}^{n} f_i(x_1) + f_{int}(X) \tag{9.5}$$

The pseudo code of the MSGP algorithm is presented in Algorithm 9.1.

9.3 Case Study

The database presented by Garzon-Roca et al. [10] was employed to compare decomposition of errors using GP and MSGP methods. The database contains experimental studies of compressive strength of masonry made of clay bricks and cement mortars. The database consists of binary inputs: (1) mortar compressive strength f_m, and (2) brick compressive strength f_b with output $f(X) = f(x_1, x_2) = f(f_m, f_b)$. Both the GP and the MSGP algorithms ran in Python [2, 13] for 100 generations and a population size of 300 to build regression models for the above-

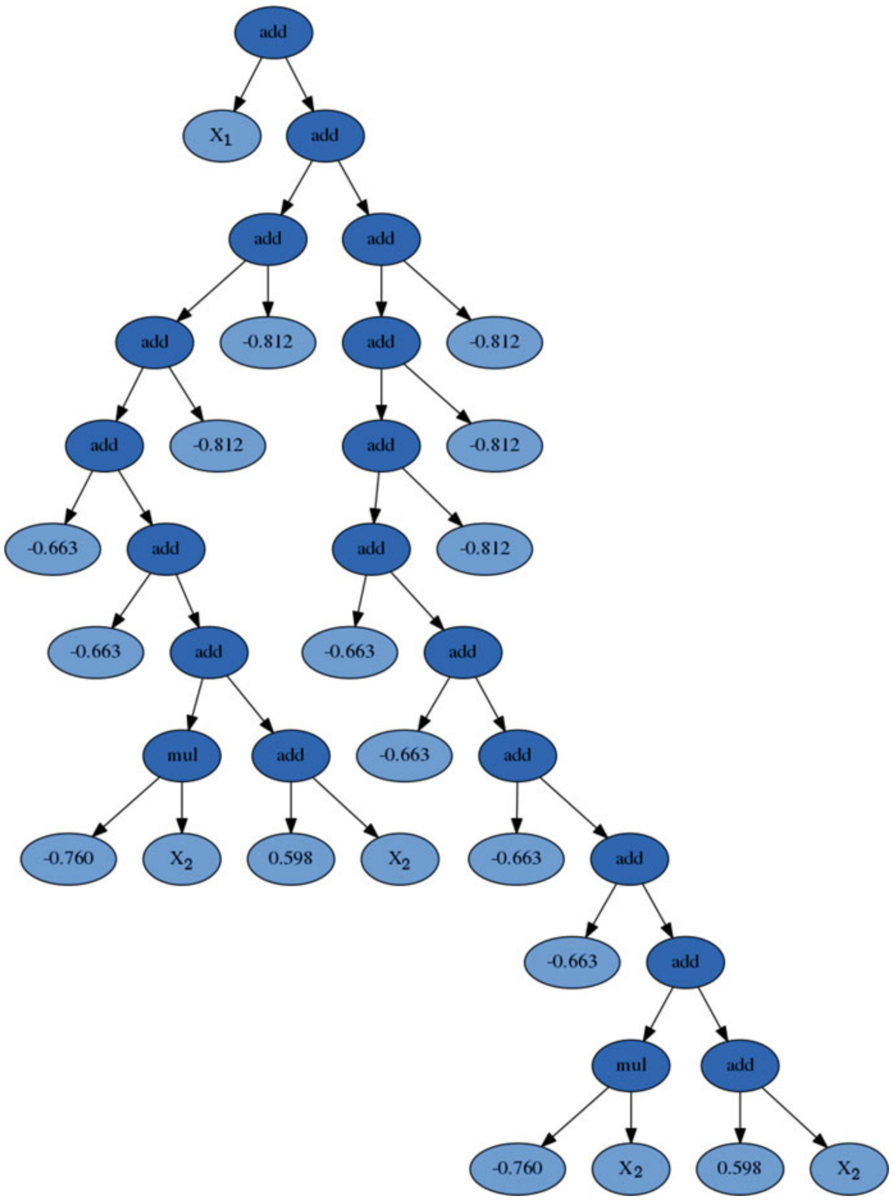

Fig. 9.2 The tree structure of the predicted solution using the GP model

mentioned database. Table 9.1 presents the details of the final parameter settings which were selected on the basis of a trial and error approach and multiple runs for the GP and MSGP algorithms. The tree structure of the resulting solution for GP is presented in Fig. 9.2. It visually presents the relation between mortar compressive

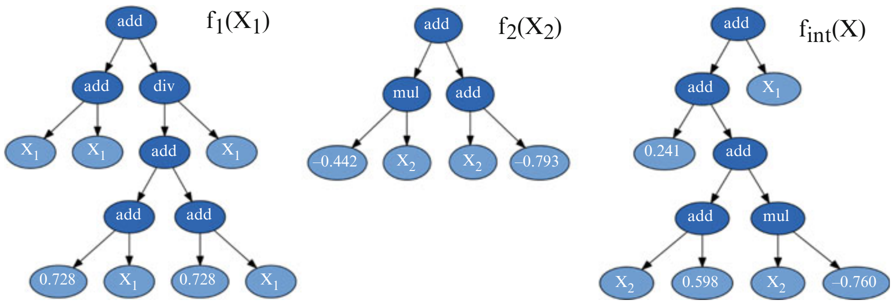

Fig. 9.3 The tree structures of the predicted solution using the MSGP model

strength ($x_1 = f_m$) and brick compressive strength ($x_2 = f_b$) to find the solution ($f(X) = f(x_1, x_2)$) using the final GP model. Additionally, Fig. 9.3 illustrates the tree structures of the solutions in the three stages of the final MSGP model.

9.4 Performance Analysis

As it is shown in Fig. 9.3, the decomposition of the errors in different stages of the MSGP method can be discussed to find out the impact of each stage in terms of error and their correlation with the outputs during the process. In this regards, the MSGP method in three different stages was considered: (1) in the first stage only the error between the actual and the predicted values based on the first variable (x_1) using GP program (f_1) was considered. (2) In the second stage, the error between the predicted values using the second input variable (x_2) and the actual values subtracted from the error calculated by (x_1), and (3) as the last stage, the sum of the errors in the first and the second stages in addition to the error calculated by the interaction function (f_{int}) were considered. In the first 10 generations, f_1 was calculated based on x_1, then from 10 to 15 generations f_2 was calculated, and from generation 15 to 25 f_{int} was used to calculate the error and R^2 score. Figure 9.4 illustrates the stages of calculating MAE through the generations. It is obvious that $f_{int}(X)$ in MSPG-II stage has just increased the accuracy of the prediction of the output values for a small amount which is infinitesimal with respect to its change in computational cost. Additionally, Fig. 9.5 also presents the R^2 score during the generations for the both GP and MSGP in the different stages. This also proves the low impact of the $f_{int}(X)$ in the MSGP$_{w-int}$ stage based on the statistical scores. More details of the calculated statistical parameters were presented in Table 9.2.

It was previously shown that a combination of minimum errors (MAE or RMSE) and $R^2 > 0.8$ lead us to a reasonable correlation between the target values and the predicted values of the models [7, 17]. To track how close the data was to the fitted regression hyperplane, R^2, the coefficient of determination for both GP and MSGP models was calculated.

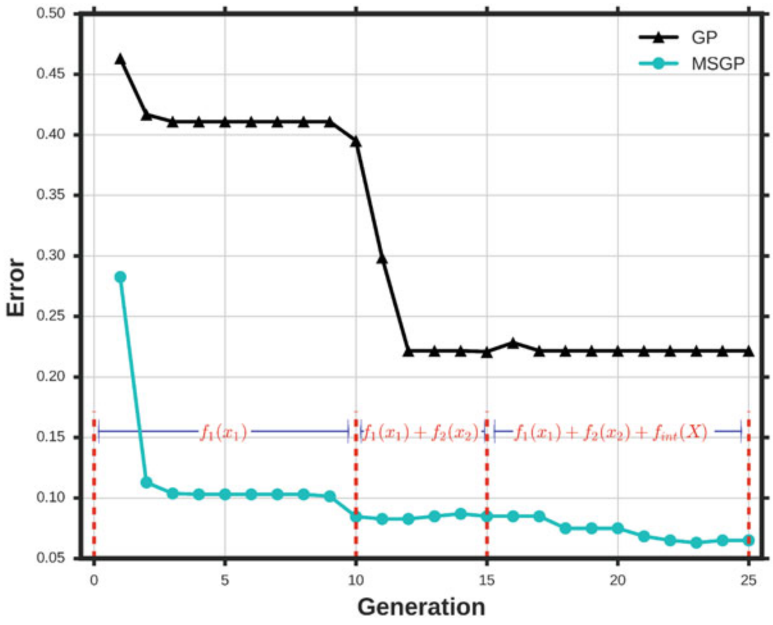

Fig. 9.4 The decomposition of the mean absolute error through the generations for the GP and the MSGP models

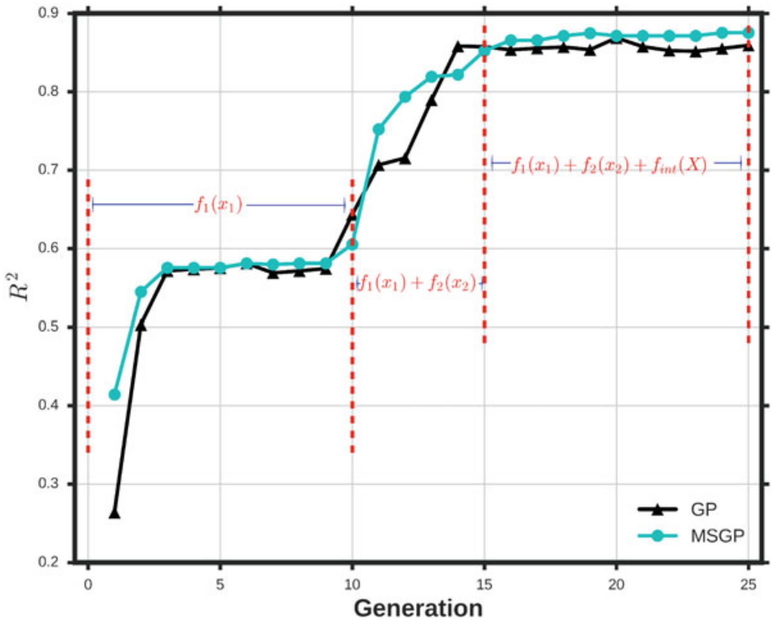

Fig. 9.5 The evolution of the coefficient of determination through the generations for the GP and the MSGP models

Table 9.2 Statistical parameters for different models

	GP	MSGP$_{wo-int}$	MSGP$_{w-int}$
R^2	0.8591	0.8581	0.8754
PI	0.1448	0.0469	0.0364
MAE	0.2215	0.0868	0.0648
$RMSE$	0.2791	0.0904	0.0706

As discussed, Table 9.2 presents the statistical scores for the GP and the MSGP models with and without $f_{int}(X)$. It also illustrates the effect of the interaction between variables. It should be noted that adding f_{int} in the MSGP$_{w-int}$ resulted in an increase of 2% in R^2 with respect to MSGP. In addition to this, the performance index (PI) was also calculated for both GP and MSGP models [8]. Considering the PI for both MSGP$_{wo-int}$ and MSGP$_{w-int}$ stages suggests the idea that the interaction function can be neglected in the multi-stage model. The MSGP strategy decreases the cost of decomposing the error by 15% and also runtime by 30%. The MSGP strategy can be employed to solve big data problems. The importance of the proposed method can be shown especially when the input numbers are high, where traditional GP method might not be a wise choice. The MSGP strategy changes the magnitude of the complexity of the problems from one N-dimensional problem to N 1-dimensional ones. It increases the efficiency by losing an infinitesimal increase in the accuracy and the correlation between the inputs and outputs.

Figure 9.6 depicts the 3-dimensional hyperplane solutions of the inputs by GP, MSGP$_{wo-int}$, and MSGP$_{w-int}$ models. It seems the interaction between variables changed the shrinkage path of the hyperplanes. As shown, standard GP always finds the best linear plane for the inputs data. Dealing with a binary problem brings the chance to illustrate both variables x_1 and x_2 and also the resulting regressed hyperplane in the same space. The most important aspect here would be how this hyperplane, without incorporating the interaction between the variables, could fit the best regression with a reasonable accuracy with respect to the actual output values.

9.5 Conclusions

This chapter discusses statistical and computational aspects of an efficient strategy called the MSGP method. It specifically focused on the decomposition of error and correlation in a multivariate nonlinear problem to reveal the capabilities of the MSGP algorithm to be applied to big data problems. The proposed method separates one N-dimensional problem into N 1-dimensional problems. To see how the MSGP performs, the performance of the MSGP model was compared with a standard GP model in the case of a nonlinear regression problem. The decomposition cost of both models during the generations was presented. Based on a high correlation in predicted values, the MSGP outperforms standard GP. These results suggest that the MSGP algorithm can be employed in various big data problems which are

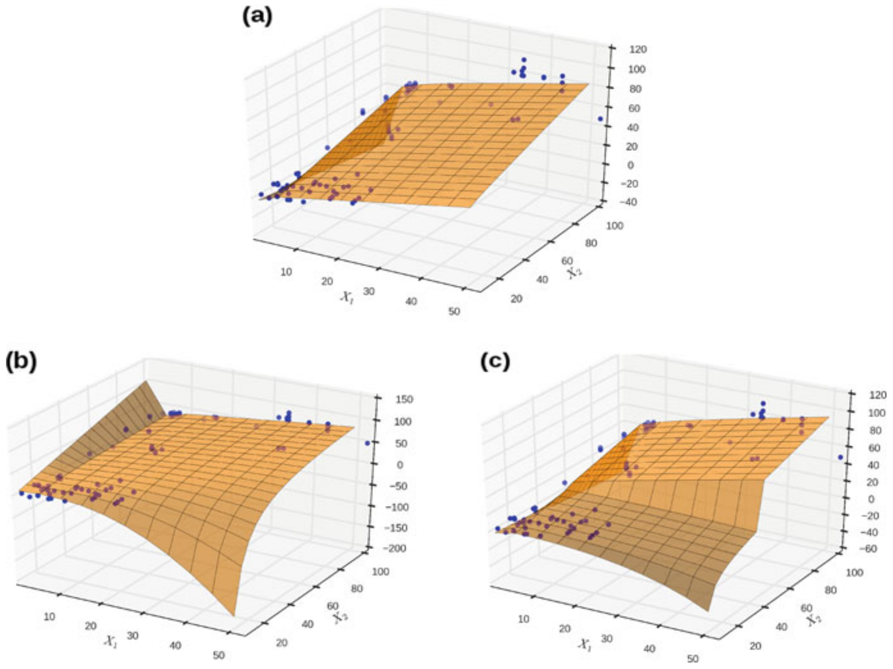

Fig. 9.6 The predicted regression hyperplanes by (**a**) GP, (**b**) MSGP$_{wo-int}$, and (**c**) MSGP$_{w-int}$ models

more difficult to solve using traditional methods due to the high computational cost. Error decreased by 15% by using the MSGP model. Additionally, the MSGP strategy reduced computational runtime by 30%. The evolution of the MAE was presented for both MSGP$_{wo-int}$ and MSGP$_{w-int}$ stages. This suggests the idea of solving the N 1-dimensional problems without considering the interactions among predictor variables. The calculated statistical scores such as R^2, and PI suggest that neglecting the interactions between the input variables causes a loss of only 1% of the correlation between the inputs and the output. This would save a reasonable amount of computational time in big data problems. To recapitulate, the MSGP method opens an innovative avenue to apply evolutionary algorithms in big data problems to overcome difficulties resulting from the big data features such as heterogeneity, noise accumulation, spurious correlation, and incidental endogeneity.

Acknowledgements This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. For valuable help in the revision of the chapter we also would like to thank Eitan Lees for critical and helpful comments on final draft.

References

1. Brameier, M.F., Banzhaf, W.: Linear genetic programming. Springer Science & Business Media (2007)
2. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
3. Fan, J., Han, F., Liu, H.: Challenges of big data analysis. National Science Review **1**(2), 293–314 (2014)
4. Gandomi, A.H., Alavi, A.H.: Multi-stage genetic programming: a new strategy to nonlinear system modeling. Information Sciences **181**(23), 5227–5239 (2011)
5. Gandomi, A.H., Alavi, A.H.: A new multi-gene genetic programming approach to non-linear system modeling. part II: geotechnical and earthquake engineering problems. Neural Computing and Applications **21**(1), 189–201 (2012)
6. Gandomi, A.H., Alavi, A.H.: A new multi-gene genetic programming approach to nonlinear system modeling. part I: materials and structural engineering problems. Neural Computing and Applications **21**(1), 171–187 (2012)
7. Gandomi, A.H., Alavi, A.H., Mirzahosseini, M.R., Nejad, F.M.: Nonlinear genetic-based models for prediction of flow number of asphalt mixtures. Journal of Materials in Civil Engineering **23**(3), 248–263 (2010)
8. Gandomi, A.H., Roke, D.A.: Assessment of artificial neural network and genetic programming as predictive tools. Advances in Engineering Software **88**, 63–72 (2015)
9. Gandomi, A.H., Sajedi, S., Kiani, B., Huang, Q.: Genetic programming for experimental big data mining: A case study on concrete creep formulation. Automation in Construction **70**, 89–97 (2016)
10. Garzón-Roca, J., Marco, C.O., Adam, J.M.: Compressive strength of masonry made of clay bricks and cement mortar: Estimation based on neural networks and fuzzy logic. Engineering Structures **48**, 21–27 (2013)
11. Iba, H., deGaris, H., Sato, T.: A numerical approach to genetic programming for system identification. Evolutionary Computation **3**(4), 417–452 (1995).
12. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT Press (1992)
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
14. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming. Lulu. com (2008)
15. Ryan, C., Collins, J., Neill, M.: Grammatical evolution: Evolving programs for an arbitrary language. In: European Conference on Genetic Programming, Paris 1998, pp. 83–96 (1998) Springer, Berlin (1998)
16. Schadt, E.E., Linderman, M.D., Sorenson, J., Lee, L., Nolan, G.P.: Cloud and heterogeneous computing solutions exist today for the emerging big data problems in biology. Nature Reviews Genetics **12**(3), 224–224 (2011)
17. Smith, G.N.: Probability and statistics in civil engineering. Collins Professional and Technical Books **244** (1986)
18. Tahmassebi, A., Gandomi, A.H.: Building energy consumption forecast using multi-objective genetic programming. Measurement **118**, 164–171 (2018)
19. Tahmassebi, A., Gandomi, A.H., McCann, I., Schulte, M.H., Schmaal, L., Goudriaan, A.E., Meyer-Bäse, A.: An evolutionary approach for fMRI big data classification. In: 2017 IEEE Congress on Evolutionary Computation (CEC) pp. 1029–1036 (2017)

20. Tahmassebi, A., Gandomi, A.H., Meyer-Bäse, A.: High performance GP-based approach for fMRI big data classification. In: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, PEARC17, pp. 57:157:4. ACM Press, New York, NY, USA (2017)
21. Wu, X., Zhu, X., Wu, G.Q., Ding, W.: Data mining with big data. IEEE transactions on knowledge and data engineering **26**(1), 97–107 (2014)
22. Zhang, B.T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation **3**(1), 17–38 (1995)