Generalisation Enhancement via Input Space Transformation: A GP Approach

Ahmed Kattan[1], Michael Kampouridis[2], and Alexandros Agapitos[3]

[1] Um Al Qura University, AI Real-World Applications Lab, Department of Computer Science, Kingdom of Saudi Arabia
ajkattan@uqu.edu.sa
[2] University of Kent, School of Computing, United Kingdom
M.Kampouridis@kent.ac.uk
Complex and adaptive systems laboratory, School of computer science and informatics, University College Dublin
alexandros.agapitos@ucd.ie

Abstract. This paper proposes a new approach to improve generalisation of standard regression techniques when there are hundreds or thousands of input variables. The input space X is composed of observational data of the form $(x_i, y(x_i)), i = 1...n$ where each x_i denotes a k-dimensional input vector of design variables and y is the response. Genetic Programming (GP) is used to transform the original input space X into a new input space $Z = (z_i, y(z_i))$ that has smaller input vector and is easier to be mapped into its corresponding responses. GP is designed to evolve a function that receives the original input vector from each x_i in the original input space as input and return a new vector z_i as an output. Each element in the newly evolved z_i vector is generated from an evolved mathematical formula that extracts statistical features from the original input space. To achieve this, we designed GP trees to produce multiple outputs. Empirical evaluation of 20 different problems revealed that the new approach is able to significantly reduce the dimensionality of the original input space and improve the performance of standard approximation models such as Kriging, Radial Basis Functions Networks, and Linear Regression, and GP (as a regression techniques). In addition, results demonstrate that the new approach is better than standard dimensionality reduction techniques such as Principle Component Analysis (PCA). Moreover, the results show that the proposed approach is able to improve the performance of standard Linear Regression and make it competitive to other stochastic regression techniques.

Key words: Genetic Programming, Symbolic Regression, Approximation Models, Surrogate, Dimensionality Reduction

1 Introduction

Science and engineering design problems oftentimes require the construction of a model $\hat{f}$ (referred to as meta-model, response surface model, or surrogate) that emulates the response of some black-box f which comes from some process. These black-box problems, i.e., whose problem class is unknown, are possibly mathematically ill-behaved (e.g., discontinuous, non-linear, non-convex). Generally, the model $f(x)$ represents some continuous quality or performance measure of a process defined by k-vector design variables $x \in X \subset \mathbf{R}^k$. In the remainder of this paper we will refer to X as the *input space*. Normally, the only insight available about the model $f(x)$ is through some discrete samples $(x_i, y(x_i)), i = 1...n$ where each x_i denotes a k-dimensional input vector of design variables and y is the response. The task here is to construct an approximation model $\hat{f(x)}$ to map any unseen $x \in X$ to its response with a reasonable accuracy.

It should be noted that reliable approximation models in the field of Machine Learning (ML) revolve around the fundamental property of generalisation. This ensures that the induced model is a concise approximation of a data-generating process and performs correctly when presented with data that has not been utilised during the learning process. To this end, it is desirable to avoid complexity of approximation models to maintain good generalisation. Thus, it is intuitively obvious that a higher number of design variables in a modelling problem will increase the complexity of objective function measuring locations of sampled variables in the input space and subsequently effect the generalisation ability. Moreover, the high number of design variables often requires more samples to build a reasonable accurate approximation model and, thus, increases the learner's complexity and may reduce its generalisation. This problem is referred to as *curse of dimensionality* [3]. To this end, many data-centric approximation methodologies in the ML literature that have been used to construct approximation models yield poor performance when the number of design variables is high.

One way to mitigate the curse of dimensionality problem is by reducing the number of design variables using some dimensionality reduction technique such as Principle Component Analysis (PCA) or Factor Analysis (FA) (e.g., see [8]). However, variables reduction is reasonable only when the significant variables are just a fraction of the overall set of variables. Variable reduction, some times, can increase the difficulty of the problem in cases where all variables have similar influence on the model response. Another way to deal with the curse of dimensionality is to construct a new input space that can be mapped to the original input space and is easer to approximate [13].

This paper proposes a model to improve the generalisation performance of standard regression models when the number of design variables is high. The main idea is to use Genetic Programming (GP) [11] to evolve a transformation function that transforms the original input space X into a new input space Z that has smaller number of variables and is easier to approximate to their corresponding responses. To this end, GP individuals (represented as trees) receives the design variables from the original input space as inputs and return a vector of outputs.[3] The evolution of the transformation function is guided by a fitness measure that drives search toward performance improvement of standard approximation models. For this task, GP is supplied with a function set that allows the extraction of statistical features from the original input space (details in Section 3).

The contribution of this paper is twofold. First, we show that it is possible to improve the generalisation of approximation models just by transforming the input space without changing anything in the approximation models themselves or in their objective functions. Second, we show that our approach can boost the performance of a simple linear regression and make it competitive to other state-of-the-art approximation techniques.

The reader's guide to the rest of the paper is as follows. Section 2 presents related work from the literature. Section 3 presents the proposed approach in details followed by experimental results and their analysis in Section 4. Finally, this paper concludes in Section 5.

2 Related Works

Dimensionality reduction techniques to mitigate the curse of dimensionality problem is a well explored topic. Many techniques have been developed and used with feature selection and classification problems (e.g., [12], [2]). However, the idea of reducing the number of design variables in the regression problems to improve generalisation of standard ML approaches is relatively little explored thus far. In this section we focus the

[3] We used a design similar to *modi GP* proposed by Zhang et. al. in [14] to allow GP trees produce multiple outputs.

review on dimensionality reduction approaches for models approximation since these are directly relevant to the work reported in this paper.

Sobester and Nair in [13] presented a GP approach for generating functions in closed analytic form that map the input space of a complex function approximation problem into one where the output is more amenable to linear regression. To achieve this, the authors used a co-evolutionary approach where multiple populations are evolved in parallel. However, the authors claimed that their results are not conclusive and they are merely serve as proof of concept. In addition, the new transformed input vector **z** has the same dimensionality as the original vector.

In [8] the authors proposed a technique based on latent variables, non-linear sensitivity analysis, and GP to manage approximation problems when the number of input variables is high. The proposed technique was tested with 340 input variable problems. The proposed approach was designed to consider problems where all input variables have similar influence on the model's output. Thus, standard variable pruning techniques are not applicable.

McConaghy [9] presented a deterministic technique, referred to as Fast Function Extraction (FFX), for solving a symbolic regression problem that achieves higher approximation accuracy than standard GP and several state-of-the-art regression techniques. Later, Icke and Bongard [5] hybridised FFX and GP to create an improved learner for symbolic regression problems. In this work, the authors showed that a hybrid deterministic/GP for symbolic regression outperforms GP alone and several state-of-the-art deterministic regression techniques alone on a set of multivariate polynomial symbolic regression tasks. The proposed approach was tested to approximate data-sets of different dimensionality, ranging from 1 to 25 dimensions.

As it can be seen, most of previous work tried to mitigate the curse of dimensionality problem by transforming the input space into a new input space. In this paper we show that it is possible to mitigate the curse of dimensionality problem and improve the generalisation of approximation models just by transforming the input space into new space that holds similar features. Unlike other works, our approach builds a transformation function for the input space based on its statistical features. This allows the transformation function to significantly reduce the number of design variables and relax the learners' performance.

3 Proposed Approach

The proposed approach uses GP as the main engine to transform the original input space into a new one. GP individuals are designed to receive the training samples from X as inputs and return a transformed samples as an output. This can be represented formally as follows: let the original input space be denoted as $X \subset \mathbf{R}^k$ where k is the dimensionality of the input samples. Normally, the input space is represented with a set of n discrete and possibly sparse samples $X = \{x_0, x_1, ..., x_n\}$. The aim is to evolve a transformation function $T(X) \Rightarrow Z$ where $Z \subset \mathbf{R}^q$ and $q < k$. The set $Z = \{z_0, ..., z_n\}$ where each z_i represent the x_i after being transformed from $X \Rightarrow Z$. The newly evolved Z set has to be easier to approximate and support the learner's generalisation. To this end, GP starts by randomly initialising a population of trees using ramped half-and-half approach [11]. We supplied GP with a function set, as illustrated in Table 1, to extract statistical features from the design variables of each sample in original input space. For each tree in the GP population, each node that holds a statistical function will be associated with a randomly selected sub-set of variables (up to k variables). For example, say $k = 10$, a tree could calculate the *Mean* function for variables $\{1, 2, 5, 6, 7\}$ while another tree (or even another node in the same tree) could calculate the *Mean* function for variables $\{9, 10\}$. We let the system picks up uniformly the number d of variables in $[1, k]$, and then uniformly choose the d variables among all variables. Once the system

Table 1. GP Function set

Function	Arity	Input	Output
+, -, /, *	2	Real Number	Real Number
Mean, Median, StD, Variance, Average Div, Min, Max	1	Randomly selected variables from each x_i	Real Number
Constants 1-6	0	N/A	Real Number

**StD* is Standard Deviation, and *Average Div* is Average Deviation

allocates a randomly selected sub-set of design variables to a node that holds a statistical function, it maintains the same sub-set for that node during its life cycle. Thus, all nodes that hold statistical functions maintain their selected sub-sets of variables after crossover or reproduction.

The next sub-section will explain the design used for GP trees to allow them to produce multiple outputs.

3.1 Trees return multiple outputs

Standard GP tree representation proposed by Koza [7] utilises the evolved functions to perform a many-to-one mapping from inputs to an output. However, some problems, such as the one presented in this paper, raise a need to evolve functions that perform a many-to-many mapping between inputs and outputs. The idea of evolving trees with multiple outputs has been proposed by Zhang et. al. in [14]. In this work the authors proposed a new representation called *modi* trees to produce multiple outputs. Here, our design is inspired by modi trees, however, the difference is that modi was presented to evolve a fixed size vector of outputs while our approach can evolve a vector of any size. Similar to modi, our tree design consists of two main types of nodes: *(a)* **standards nodes** that pass their outputs to their parents, and *(b)* **outputs nodes** that append their output to a vector associated with the tree. As illustrated in Figure 1, for each tree, we let the system randomly selects some nodes and label them as output nodes. Thus, the size of the output vector of each tree is equal to the number of its output nodes. Also, once the system labels a node in any tree as an output node, it maintains its type during its life cycle (i.e., after crossover or reproduction). When evaluating trees, the system ensures to maintain the same traverse order, thus, the same tree will always produce the same output vector.

Using this representation allows the outputs of the evolved z_i vectors to be generated from different sub-trees or leaf nodes. Hence, any element in the generated output vector can be the result of a simple statistical feature for a sub-set of the design variables or the result of a linear function that combines several statistical features.

3.2 Fitness measure

As mentioned previously, the aim of the transformation function is to improve the generalisability of standard learners (or model approximation techniques). This is a challenging problem because the fitness measure needs to be aware of the generalisation level induced by the transformed space. In addition, the evolved transformation function has to be applicable to several learners without making any prior assumptions about which learner will be used to solve the approximation problem. One simple way to test the quality of the transformation function is to use the transformed input samples to train a single learner, e.g., Radial Basis Function Networks (RBFN), and then test this learner with unseen data (i.e., validation set). Although the simplicity of this idea is intuitively appealing, one problem would lie in the selection of the learner that will be used as

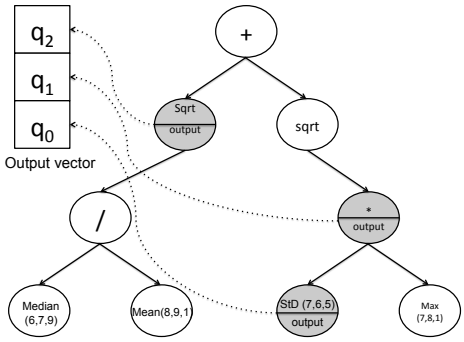

Fig 1. GP tree representation to produce multiple outputs inspired by modi.

fitness measure. As demonstrated in preliminary experiments, GP tunes the input space quickly in such a way to allow the learner to over-fit samples of the transformed input space. Thus, the learner loses its generalisation property. In fact, when we used prediction errors of RBFN as fitness measure, in preliminary experiments, we found that GP evolves a transformation functions that allows RBFN to over-fit the training set easily. Hence, the fitness value keeps improving, which gives an indication that RBFN is doing well on the transformed input space. However, when testing the RBFN on a transformed validation set results show very poor performance.

To avoid pathologies of this kind, another idea to evaluate the quality of the evolved transformation functions is to use multiple learners and solve the problem as multi-objectives optimisation, where the system is trying to reduce the prediction error of all learners simultaneously. Then the best evolved transformation function, at each generation, is evaluated against a validation set. In preliminary experiments, we found that this idea worked well to some extent, however, its main drawback is emphasised in the large computational cost required to train each learner using the transformed samples induced by each individual in the GP population. Hence, we reduced the number of learners (used as a fitness measure) gradually and explored different permutations of learners to balance between computational cost and solutions' quality.

After several experiments, based on trial and error, we used two learners as a fitness measure; RBFN and Linear Regression (LR). Generally, RBFN shows its best performance when the landscape of input space is smooth [6]. Thus, using prediction errors of RBFN as fitness measure, in principle, will guide GP to evolve smooth transformed input space in such a way to reduce the prediction errors of RBFN. However, if the transformed training samples became congregated in a small area at the transformed input space then it is most likely that RBFN will over-fit the training samples. Here, we also use the LR model to mitigate this problem. In LR, data are modelled using linear predictor functions. LR shows its best performance when the input samples are linearly correlated with their outputs. Also, LR performance significantly decreases when the number of design variables increases. With these properties, LR can encourage GP to linearly align the transformed samples to their corresponding outputs and reduce the number of input variables. To this end, LR and RBFN are selected to guide the fitness measure of GP. In addition, they both are relatively fast algorithms which will result in a reasonable computational cost when evaluating GP individuals. More formally, the fitness measure can be denoted as follows:

$$Fitness = \frac{\sum_{i=0}^{n} |LR(z_i) - y_i|}{n} + \frac{\sum_{i=0}^{n} |RBFN(z_i) - y_i|}{n} \quad (1)$$

where $LR(z_i)$ and $RBFN(z_i)$ represent the predictions of LR and RBFN given the transformed point z_i. The n is the number of the transformed input samples and y_i is the i^{th} output.

GP evolves individuals as described in Section 3.1 where each individual produces a $z_i \in Z$ output vector for each input sample $x_i \in X$. Remember that we assume each x_i denotes a k-dimensional input vector of design variables. The set $Z = \{z_0, ..., z_i\}$ is used to train RBFN and LR using 2-fold cross-validation technique. The prediction error is calculated as described in Equation 1 to rank individuals. The best individual of each generation is further tested with an unseen transformed validation set. The best individual across the whole run (that produced the best performance on the validation set) is used as the final transformation function that can be used to transform the input space.

4 Experiments and Analysis

4.1 Experimental Settings

A set of experiments have been conducted to evaluate the proposed approach. We tested the effects of the transformation function on four regression models, namely, RBFN, Kriging, LR, and GP (used as a regression model). These models were selected because they are some of the most important techniques in the literature. In addition, to compare the proposed approach against standard dimensionality reduction technique we included Principle Component Analysis (PCA) [1] in the experiments.

Experiments included the following 5 benchmark functions; *Rastrigin, Schwefel, Michalewicz, Sphere,* and *Dixon & Price* [10]. For each test function, we trained all approximation models to approximate the given function when the number of variables is 100, 500, 700 and 1000. The total number of test problems is 20 (i.e., 5 test functions $\times 4$ different variables sizes). For all test problems, we randomly generated three disjoint sets; a training set of 100 points, a validation set of 50 points, and a testing set of 150 points from the interval $[-5, 5]$. All techniques have been compared based on the average of absolute errors on the testing set.

For each function-variables combination, each approximation model has been tested three times; without and with PCA and our proposed approach (we will call it the Z set). In the experiments, some models are deterministic so we tested them only once with each problem. These, in particular, are RBFN, Kriging, and LR, with and without PCA. However, because the generation of the transformation function is based on an evolutionary process, we evolved 30 different transformation functions and tested them with each approximation model for each problem and reported the *mean, median, best,* and *standard deviation*. To evolve the transformation function, we used generational GP with the following settings; population size is 100, number of generations 100, maximum allowed size for trees is 300 nodes, elitism rate 0.01, crossover and mutation rates 0.7 and 0.3, respectively.

For the GP engine that has been used to solve the approximation problems, we used the same settings described previously. To assure fair comparison, we tested standard GP with and without PCA in 30 different runs and reported the same results.

4.2 Results

Tables 2, 3, and 4 summarise the results of 1200 GP runs and 60 RBFN, Kriging, and LR runs. In Table 2, it is clear that the Z set has improved all approximation techniques to obtain the best error in all four variables sizes (denoted by **bold** fonts in any column with a title starting with "Z set"). In addition, the Z set improved the GP performance in terms of mean in 7 out of 8 test cases and in terms of median in 5 cases (see again the numbers in **bold** fonts in the column entitled "Z set+SGP"). We also, noted that the Z set improved the performance of LR significantly by several orders of magnitude. In fact, LR obtained the best overall approximation for 6 out of 8 cases (denoted by

underlined fonts). These are remarkable results given that LR simply uses a linear function to make predictions. The results suggest that the evolved transformation function has aligned the transformed input variables to be linearly correlated with their outputs.

Similarly, in Table 3, the Z set has improved all approximation techniques in terms of best results. Also, the Z set improved the GP performance in terms of mean in 7 out of 8 test cases and in terms of median in 4 cases. The LR obtained the best overall approximation for 6 out of 8 cases. Finally, in Table 4 results follow the same pattern[4]. The Z set again leads to consistently improved results in terms of best values. It also improved the mean and median results in all 4 cases. Lastly, the combination of the Z set with LR was again the champion, having the best overall approximation in all 4 cases.

To further verify the significance of the non-deterministic results on the GP, we used the non-parametric Friedman test to rank the three algorithms tested, namely SGP, Z set+SGP, and PCA+SGP. As we can observe from Table 5, the SGP approximation that uses the Z set was ranked first in 16 out of the 20 test cases; in addition, Holm's post-hoc test [4] showed that 12 out of these 16 first ranking were statistically significant at 5% or 10% level. It should also be noted that none of the other two algorithms (SGP, PCA+SGP) has managed to be ranked first at a statistical significance level of 5%. This once again demonstrates the improvement brought by the Z set to GP with these benchmark functions.

Overall, results show that the transformed input space has managed to improve the generalisation of all approximation techniques in the comparison. Thus, when the evolved Z set is applied, we can expect to have an improvement in the approximation error. In addition, results also show that our approach is better than a standard dimensionality reduction technique, such as PCA. Moreover, the transformed input space has significantly improved the LR in most of the test cases and make it competitive to other stochastic approximation techniques. In fact, LR has outperformed all of its competitors in most of the cases and it is not too far behind when it loses the comparison.

To have a closer look at LR improvement with the Z set, Figure 3 depicts the approximation of Sphere function (i.e., function 4), with 2 variables, of all models included in the comparison. It is interesting to visually see that LR can accurately approximate a non-linear function only by transforming the input space. In addition to this, the LR with the Z set approximation had the lowest error (0.0445021) among all 12 algorithms.

Despite the good improvements obtained by using the Z set, it is fair to report that the main disadvantage of the proposed approach that it requires extra computational cost and time to transform the input space. Also, in some cases the improvements are not significant and, thus, can not justify the extra costs. However, as demonstrated by the results, in some cases the margins of improvement can be several orders of magnitude (e.g., Functions 3 and 4) which justifies the extra computational costs in return of higher accuracy. Lastly, it is also worth noting that the results seem to worsen as the number of dimensions increases.

One last contribution of our work is the significant dimensionality reduction. Table 6 illustrates these reductions in each test problem. As can be seen, the proposed approach has generated a new input that has more than 50% smaller number of design variables. We believe the great amount of reduction is largely attributed to the fact that the new input space is based on statistical features extracted from the original space. Thus, in a sense, each design variable in the new transformed space is a result of several variables from the original space. To this end, both input spaces (the original and transformed) have similar statistical features.

[4] Due to space limitation we did not report the results of the RBFN (with and without PCA and the Z set) in Table 4. However, in our experiments we found that the Z set has also improved RBFN.

Table 2. Summary results for functions 1 and 2

	SGP	Z set+SGP	PCA+SGP	RBFN	Z set+RBFN	PCA+RBFN	Kriging	Z set+Kriging	PCA+Kriging	LR	Z set+LR	PCA+LR
Function 1 (Rastrigin) Dimensions 100												
Mean	7524868.16	**73.67**	159.83		90.10			73.67			76.55	
Best	75.37	**73.52**	73.65	82.19	**73.72**	82.48	73.68	**73.52**	73.68	72740.90	**59.76**	
Median	90.81	**73.68**	91.29		88.68			73.68			70.81	
Function 1 (Rastrigin) Dimensions 500												
Mean	1001.54	**209.01**	916.14		210.75			184.92			22618.21	
Best	174.04	**130.82**	169.12	185.74	**162.89**	185.82	180.56	**176.93**	180.56	1.54E+07	**148.69**	
Median	213.37	**182.00**	215.46		209.07			180.56			167.80	
Function 1 (Rastrigin) Dimensions 700												
Mean	2666.25	**3824.53**	572.18		269.65			207.42			102342.89	
Best	220.82	**174.74**	208.36	216.14	**211.43**	216.15	207.39	**206.43**	207.39	4.07E+07	**176.47**	
Median	256.34	252.04	**240.69**		262.31			207.39			226.80	
Function 1 (Rastrigin) Dimensions 1000												
Mean	2624.30	2892.22	**411.94**		276.75			241.54			135462.40	
Best	227.57	**195.21**	232.02	224.87	**213.65**	224.90	**226.37**	**226.37**	**226.37**	3.42E+08	**184.20**	
Median	305.58	285.32	**273.02**		256.83			226.37			243.05	
Function 2 (Schwefel) Dimensions 100												
Mean	13072.65	**4152.98**	9204.97		2927.30			21681.90			2384.71	
Best	3483.68	**2083.80**	3471.49	4116.45	**2408.64**	4100.54	3442.97	**3413.25**	3442.97	4.43E+06	**1808.15**	1.04E+06
Median	4198.08	**2832.10**	4029.32		2880.16			3460.26			2325.13	
Function 2 (Schwefel) Dimensions 500												
Mean	1184267.92	**123639.32**	1378451.65		36678.20			75874.60			28633.29	
Best	38126.30	**22711.60**	37872.80	39841.60	**26812.60**	39844.80	38258.00	**37875.30**	38258.00	1.74E+08	**22197.20**	3.74E+08
Median	52331.65	**36961.20**	48515.85		36363.70			38275.65			27407.05	
Function 2 (Schwefel) Dimensions 700												
Mean	18921991.63	**214283.00**	1913215.34		51673.54			244632.34			3700214.70	
Best	65713.60	**31523.20**	67554.90	65553.50	**37176.90**	65583.20	64037.20	**63658.60**	2673180.00	5.70E+08	**30852.70**	1.66E+09
Median	89011.40	**61952.60**	83371.25		48647.55			64182.30			39201.25	
Function 2 (Schwefel) Dimensions 1000												
Mean	7444259.17	**1186439.44**	12553754.33		105327.99			368505225.77			24271339.30	
Best	116543.00	**68884.00**	115053.00	119038.00	**69365.30**	119056.00	117484.00	**116987.00**	117484.00	1.21E+10	**61209.80**	2.40E+09
Median	4078830.00	181929.00	**139770.50**		97153.70			117484.00			72801.55	

* **Bold** numbers are the lowest in each group and underlined numbers are the lowest in all groups

Table 3. Summary results for functions 3 and 4

	SGP	Z set+SGP	PCA+SGP	RBFN	Z set+RBFN	PCA+RBFN	Kriging	Z set+Kriging	PCA+Kriging	LR	Z set+LR	PCA+LR
Function 3 (Michalewicz) Dimensions 100												
Mean	1.07E+04	**5.26E+06**	3.99E+03		6.07			184.12			3.46	
Best	19.18	**2.89**	20.61	15.74	**2.65**	15.72	20.74	**20.62**	20.74	9.47E+05	**1.39**	6.28E+05
Median	28.66	43.72	**24.89**		5.04			25.32			2.05	
Function 3 (Michalewicz) Dimensions 500												
Mean	1.89E+04	2.20E+05	**1.52E+04**		5877433.82			5.88E+06			10.42	
Best	48.89	**9.15**	48.79	48.55	**49.33**	48.58	49.40	**49.33**	49.40	4.89E+08	**4.13**	9.00E+07
Median	55.99	62.00	**54.78**		110.23			110.23			7.46	
Function 3 (Michalewicz) Dimensions 700												
Mean	8.71E+04	**2.58E+04**	5.60E+04		16.42			1.91E+07			9.35	
Best	52.41	**9.12**	52.11	50.91	**7.04**	50.89	52.43	**49.23**	52.43	1.04E+08	**4.73**	2.54E+08
Median	133.01	1625.70	**62.03**		14.93			168.26			7.48	
Function 3 (Michalewicz) Dimensions 1000												
Mean	1.34E+05	**1.05E+05**	6.04E+04		32.20			2.00E+08			6.50E+04	
Best	69.84	**9.11**	69.65	67.61	**9.87**	67.64	69.13	**68.92**	3897.60	1.59E+08	**6.80**	4.20E+08
Median	296.60	235.04	**79.51**		23.20			98.99			14.07	
Function 4 (Sphere) Dimensions 100												
Mean	61.06	**22.79**	61.58		11.28			1.56E+08			11.17	
Best	49.79	**9.57**	48.67	55.65	**0.02**	56.51	48.98	**48.09**	48.98	3.10E+04	**6.29**	3.10E+04
Median	58.41	**16.07**	54.63		12.52			94.32			10.26	
Function 4 (Sphere) Dimensions 500												
Mean	503.80	**132.65**	172.44		27.80			1.31E+04			18.74	
Best	144.81	**11.74**	145.18	146.93	**0.21**	147.12	146.31	**145.52**	146.31	1.71E+07	**8.84**	1.44E+06
Median	157.77	**71.33**	154.52		29.12			146.58			19.70	
Function 4 (Sphere) Dimensions 700												
Mean	653.16	**73.12**	256.48		32.56			4.41E+10			16.56	
Best	142.39	**13.00**	144.92	148.94	**8.56**	148.99	147.31	**143.35**	8139.38	1.46E+06	**7.08**	2.63E+06
Median	178.88	**50.89**	160.93		27.89			5718.59			15.57	
Function 4 (Sphere) Dimensions 500												
Mean	2420.14	**83.71**	572.57		34.72			2.48E+10			18.32	
Best	172.42	**9.74**	180.50	176.93	**9.08**	176.93	177.25	**175.56**	177.25	4.50E+07	**6.83**	1.53E+07
Median	233.90	**71.13**	219.29		32.72			1.33E+04			18.99	

* **Bold** numbers are the lowest in each group and underlined numbers are the lowest in all groups

Table 4. Summary results for function 5 Function 5 (Dixson& Price)

	SGP	Z set+SGP	PCA+SGP	Kriging	Z set+ Kriging	PCA+ Kriging	LR	Z set+ LR	PCA+ LR
Dimensions 100									
Mean	4.9E+07	**7.4E+05**	2.9E+06		1.2E+05			2.2E+05	
Best	5.3E+05	**3.3E+05**	5.3E+05	4.3E+05	**2.6E+05**	4.3E+05	2.0E+07	**1.8E+05**	4.2E+06
Median	6.0E+05	**5.0E+05**	6.2E+05		2.3E+05			2.16E+05	
Dimensions 500									
Mean	5.8E+07	**2.8E+07**	3.9E+07		2.9E+06			2.4E+06	
Best	6.1E+06	**3.7E+06**	6.3E+06	3.72E+06	**2.3E+06**	3.7E+06	2.4E+10	**2.1E+06**	1.7E+10
Median	7.2E+06	**6.3E+06**	7.1E+06		2.9E+06			2.3E+06	
Dimensions 700									
Mean	4.7E+07	**4.3E+07**	6.3E+07		5.3E+06			9.1E+06	
Best	1.0E+07	**6.2E+06**	1.0E+07	6.5E+06	**4.1E+06**	6.5E+06	2.6E+10	**3.65E+06**	7.41E+10
Median	1.21E+07	**9.1E+06**	1.4E+07		5.0E+06			4.07E+06	
Dimensions 1000									
Mean	3.6E+08	**1.1E+08**	2.1E+08		9.7E+06			7.10E+06	
Best	1.7E+07	**1.0E+07**	1.7E+07	9.7E+06	**7.0E+06**	9.7E+06	1.2E+12	**6.0E+06**	1.2E+11
Median	1.8E+08	**1.7E+07**	3.4E+07		9.5E+06			6.87E+06	

* **Bold** numbers are the lowest in each group and underlined numbers are the lowest in all groups

Table 5. Friedman statistical significance test. Values that are in **bold** font denote that the respective algorithm's ranking is statistically better than one other algorithm (at 5% significance level). Values that are both in **bold** font and underlined denote that the algorithm has a statistically better ranking than both of the other two algorithms (at 5% significance level). Lastly, when a value has a star (*) next to it, this means that the respective algorithm has a statistically significant ranking at 10% level.

	Function	*F1*	*F2*	*F3*	*F4*	*F5*
Dimension	Algorithm	Ranking				
100	SGP	2.3	2.56	2.13	2.6	2.16
	Z set+SGP	**1.63***	**1.23**	2	**1.06**	**1.36**
	GP+PCA	2.06	2.2	1.86	2.33	2.46
500	SGP	2.2	2.2	2.16	2.53	2.16
	Z set+SGP	**1.53**	1.73	1.76	**1.2**	1.7
	GP+PCA	2.26	2.06	2.06	2.26	2.13
700	SGP	2.23	2.23	2.06	2.4	2.20
	Z set+SGP	2	**1.33**	1.93	**1.2**	**1.40**
	GP+PCA	1.76	2.43	2	2.4	2.40
1000	SGP	2.23	2.18	2.36	2.46	2.40
	Z set+SGP	2.03	1.96	1.8*	**1.1**	**1.43**
	GP+PCA	1.73	1.85	1.83	2.43	2.16

However, an observation one could make is that the number of dimensions in the new input space does not seem to be correlated with the number of dimensions from the original input space; in fact, while the mean number of variables varies from 10.27 (F4) to 23.30 when Dimensions = 100, this mean range only slightly increases for Dimensions = 1000 (11.83 (F4) to 31.20 (F1)). This is very interesting and it could be an explanation as to why the approximation results are poorer for higher number of dimensions. As explained earlier in Section 3.1, our multiple-outputs GP approach can evolve an output vector of any size. However, as we see in practice this size is not proportionate to the size of the original dimension of the X vector. It would thus be worth investigating in a future work if approximation results can be improved when the number of variables in the transformed input space is higher.

5 Conclusions

To summarise, this paper proposed a new approach to improve generalisation of standard regression techniques when dealing with hundreds or thousands of input variables. We used GP to transform the original input space X into a new input space

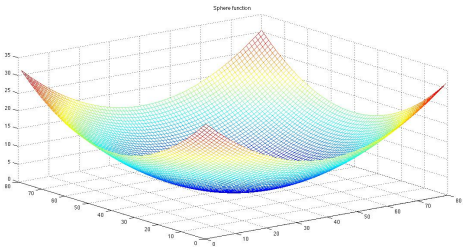

Fig 2. The original Function 4 (Sphere function).

Table 6. Summary of dimensions produced by the evolved Z set. The table summaries the mean results of 30 independent runs for each test problem.

Dimensions	*F1*	*F2*	*F3*	*F4*	*F5*
100	14.97	14.27	12.27	10.27	23.30
500	21.97	14.43	14.37	12.77	25.20
700	33.67	17.30	12.73	11.23	25.30
1000	31.20	26.63	16.60	11.83	23.26

$Z = (z_i, y(z_i))$, where Z has a smaller input vector and is thus easier to be mapped. We tested the effectiveness of our proposed approach over 5 different functions and over 4 different dimensionality sizes. Results over the above 20 problems showed that our approach leads to a remarkable dimensionality reduction of the original input space, thus making the problem at hand a less complex one. Furthermore, the transformed input space was able to lead to consistently improved performance of the standard approximation models tested in this paper, i.e. Kriging, RBFN, Linear Regression and GP. Moreover, our findings also demonstrated that our approach consistently outperforms a standard dimensionality reduction technique, such as the Principle Component Analysis. Lastly, another important result was that our proposed approach was able to significantly improve the performance of the standard Linear Regression, and actually make it the best performing technique in the majority of the cases tested in this paper.

A disadvantage of the proposed approach is that it requires extra computational cost to evolve a transformation function. This cost does not bring a guarantee that the improvements margins will be significant. However, as demonstrated by the results, in most cases the approximation improvements are significant thus justify the extra computational cost.

For future work, we will explore options to reduce the computational cost of the evolutionary process. Also, we will study the distribution of the transformed inputs on the new space. Moreover, we would like to test the approach with real-world problems.

References

1. Bishop, C.M., Nasrabadi, N.M.: Pattern recognition and machine learning, vol. 1. springer New York (2006)
2. Estébanez, C., Aler, R., Valls, J.M.: Genetic programming based data projections for classification tasks. World Academy of Science, Engineering and Technology (2005)
3. Forrester, A., Sóbester, A., Keane, A.: Engineering design via surrogate modelling: a practical guide. John Wiley & Sons (2008)
4. Garca, S., Herrera, F.: An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. Journal of Machine Learning Research 9(66), 2677–2694 (2008)
5. Icke, I., Bongard, J.: Improving genetic programming based symbolic regression using deterministic machine learning. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. pp. 1763–1770 (2013)

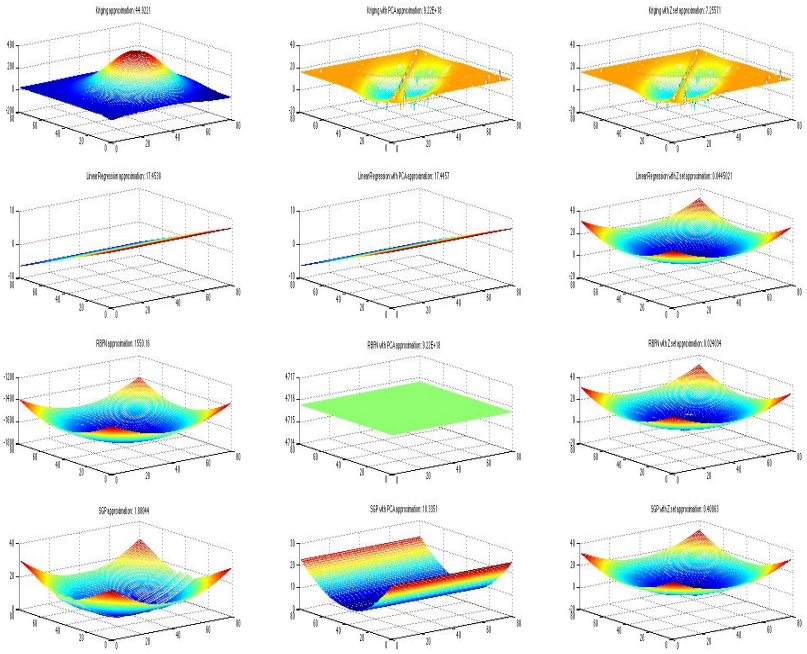

Fig 3. Approximation of Sphere function (Function 4) with 2 variables. Each model (Kriging-1st row, Linear Regression-2nd row, RBFN-3rd row, SGP-4th row) was tested without (left) and with PCA (middle) and the Z set (right).

6. Kattan, A., Galvan, E.: Evolving radial basis function networks via gp for estimating fitness values using surrogate models. In: Evolutionary Computation (CEC), 2012 IEEE Congress on. pp. 1–7 (2012)
7. Koza, J.R.: Genetic Programming: vol. 1, On the programming of computers by means of natural selection, vol. 1. MIT press (1992)
8. McConaghy, T.: Latent variable symbolic regression for high-dimensional inputs. In: Genetic Programming Theory and Practice VII, pp. 103–118. Springer (2010)
9. McConaghy, T.: Ffx: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, pp. 235–260. Springer (2011)
10. Molga, M., Smutnick, C.: Test functions for optimization needs. Test functions for optimization needs (2005)
11. Poli, R., Langdon, W.W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming. Lulu. com (2008)
12. Smits, G., Kordon, A., Vladislavleva, K., Jordaan, E., Kotanchek, M.: Variable selection in industrial datasets using pareto genetic programming. In: Yu, T., Riolo, R.L., Worzel, B. (eds.) Genetic Programming Theory and Practice III, Genetic Programming, vol. 9, chap. 6, pp. 79–92. Springer, Ann Arbor (12-14 May 2005)
13. Sobester, A., Nair, P., Keane, A.: Evolving intervening variables for response surface approximations. In: Proceedings of the 10th AIAA/ISSMO multi-disciplinary analysis and optimization conference, pp. 1–12. American Institute of Aeronautics and Astronautics (2004), `http://eprints.soton.ac.uk/22962/`, aIAA 2004-4379
14. Zhang, Y., Zhang, M.: A multiple-output program tree structure in genetic programming. In: Mckay, R.I., Cho, S.B. (eds.) Proceedings of The Second Asian-Pacific Workshop on Genetic Programming. p. 12pp. Cairns, Australia (6-7 December 2004), `http://www.mcs.vuw.ac.nz/~mengjie/papers/yun-meng-apwgp04.pdf`