GP Made Faster with Semantic Surrogate Modelling

AHMED KATTAN ajkattan@uqu.edu.sa
Computer Science Department, AI Real-World Applications Lab, UQU, Saudi Arabia

ALEXANDROS AGAPITOS alexandros.agapitos@ucd.ie
Complex and adaptive systems laboratory, School of Computer Science and Informatics, University College Dublin

YEW-SOON ONG asysong@ntu.edu.sg
School of Computer Engineering, Nanyang Technological University, Singapore

ATEQ A. ALGHAMEDI ateq@kau.edu.sa
Department of Statistics, Faculty of Science, KAU, Jeddah 21589, Saudi Arabia

MICHAEL O'NEILL m.oneill@ucd.ie
Complex and adaptive systems laboratory, School of Business, University College Dublin

Abstract.
Genetic Programming (GP) is known to be expensive in cases where the fitness evaluation is computationally demanding, i.e., object detection, programmatic compression, image processing applications. The paper introduces a method that reduces the amount of fitness evaluations that are required to obtain good solutions. We consider the supervised learning setting, where a training set of input vectors are collectively mapped to a vector of outputs, and then a loss function is used to map the vector of outputs to a scalar fitness value. Saving of fitness evaluations is achieved through the use of two components. The first component is surrogate model that predicts trees output for a particular input vector x_i based on the similarity between x_i and other input vectors in the training set for which the candidate solution has been already evaluated with. The second component, is a simple linear equation to control the size of a sub-training set that is used to train GP trees. This linear equation allows the size of the sub-training set to dynamically increase or decrease based on the status of the search. The proposed method referred to as *SSGP*. Empirical results in 17 different problems, from three different categories, demonstrate that SSGP is able to obtain solutions of similar quality with those obtained using several benchmark GP systems, but with a much smaller computation time. The simplicity of the proposed method and the ease of its implementation is one of the most appealing aspects of its future utility.

Keywords: Genetic Programming, Surrogate modelling, K-NN, Symbolic Regression, Classification, Time-series forecasting

1. Introduction

Genetic Programming (GP) is a powerful model induction method. We consider model induction to be synonymous to the general function estimation problem. One is given a set of training examples $\{x_i, y_i\}$, $i = \{1, ..., N\}$, where y is the response variable and **x** $\in \mathbb{R}^d$ is a vector of explanatory variables. The goal is to find a function $F^*(\mathbf{x})$ that maps **x** to y, such that over the joint distribution $P(\mathbf{x}, y)$ the expected value of a loss function $L(y, F(\mathbf{x}))$ is minimised:

$$F^*(\mathbf{x}) = \underset{F(\mathbf{x})}{\arg\min} \mathbb{E}_{x,y}[L(y, F(\mathbf{x}))] \quad (1)$$

Much of the power and generality of GP rises from its non-parametric nature. GP employs a variable expression-tree model representation that makes no assumption about the classes of models that will be relevant to the problem at hand, and places the least possible constraints on the space of models that are allowed to be explored throughout an evolutionary run. The trial-and-error search regime comes with the cost of interpreting a significant number of program structures for quantifying the merit of candidate solutions. In case of computationally expensive problems, like for example those of object detection, image processing, and DNA clustering, fitness evaluation can present a serious bottleneck to the evolutionary process in terms of time inefficiency. In this work, our interest is to retain the appeal of GP algorithms, which can handle challenging problems with high-quality designs at enhanced computational efficiency.

Surrogate Models (SMs) [4], also known as response surface models, are often used to reduce the cost of expensive objective functions. SMs are approximation models, that mimic the behaviour of the simulation model as closely as possible while being fast surrogates for time-consuming computer simulations. An SM is an easily evaluated mathematical model that approximates an expensive objective function as precisely as possible. The inside knowledge of the objective function is not necessary to be known or to be used for the construction of an SM. Normally, SMs are built using solely discrete evaluations of the expensive objective function. From an abstract point of view, SMs are black boxes that map inputs to their corresponding outputs using a training set and generalise this knowledge on unseen inputs in order to predict their corresponding outputs. Usually, this requires SMs' predictions to be based on some distance measure between points in the training set in order to make a reasonable prediction.

This paper proposes a novel method to build surrogate models in GP for saving upon fitness evaluations. We consider the supervised learning setting, where a training set of input vectors are collectively mapped to a vector of outputs, and then a loss function is used to map the vector of outputs to a scalar fitness value. Saving of fitness evaluations is achieved through the use K-Nearest Neighbour (KNN) surrogate [1].

In standard GP, all input vectors need to be explicitly mapped to an element of the output vector via program execution. However, the proposed method randomly chooses a sub-training set every generation (where the size of the sub-training is dynamic). Thus, only a portion of the examples that are contained in the original training set will be evaluated. The new method saves on the rest program executions by building a surrogate model that is able to predict program semantics for a particular input vector based on the similarity of this input vector with those vectors that were evaluated using standard program execution. In order to compensate the cost of a potentially complex model of program semantics we employ the very simple and non-parametric KNN averaging rule. Once the semantics have been predicted, these are mapped to a fitness value through the use of the loss function that takes the form of average mean of absolute errors. The proposed approach is referred to as *Semantic Surrogate GP* (SSGP).

The reader's guide to the rest of the article is as follows: Section 2 presents background literature on surrogate modelling. Section 3 introduces SSGP. Section 4 presents the empirical study using the proposed method. Section 5 concludes and outlines directions of future work.

2. Related Work

2.1. Surrogate Models for Vectorial Representations

The general consensus on surrogate-assisted evolutionary frameworks is that the efficiency of the search process can be improved by replacing, as often as possible, calls to the costly objective functions, with surrogates that are deemed to be less costly to build and compute. In this manner, the overall computational burden of the evolutionary search can be greatly reduced, since the effort required to build the surrogates and use them is much lower than those in the traditional approach that directly couples the evolutionary algorithm with the costly objective functions [3][27] [20] [9] [26].

Early approaches have focused on building global surrogates [27] that attempts to model the complete problem fitness landscape. However, due to the effects of the curse of dimensionality [2], many have turned to local surrogate models [6][21] or their synergies [30] [29] or ensembles. The use of domain knowledge including gradient information and physics-based models [11] [17] to improve the prediction accuracy of the surrogates have also been considered. The use of more than one surrogate, for example ensemble and smoothing surrogate models, and with preference for surrogates that generate search improvements over prediction accuracy was also considered in both single and multi-objective optimisation [16].

The early work in Ong et al. [21] was among the first to have successfully proposed the use of a local surrogate embedded in a gradient based trust-region framework with Genetic Algorithm (GA). The model uses GA to generate a population of individuals and rank them with a real-valued function. A gradient-based local search is then performed on the Surrogate Model (SM) to find new promising solutions. Both the GA and the local search are alternatively used under a trust-region framework until the optimum is found, in the spirit of Lamarckian learning. The trust-region framework is used to guarantee that the search will converge to the optimal of the original problem despite the use of surrogates. Subsequently, Lian et al. [15] also investigated the enhancement of standard GA using local surrogate search to expedite its convergence.

Lim et al. [16] presented a generalised surrogate-assisted evolutionary framework for the optimisation of problems that are computationally expensive to evaluate. The authors introduced the idea of employing multiple on-line local SMs that are constructed using data points that lie in the vicinity of an initial guess. Improved solutions generated by local search that considers the ‘Curse and Bless of uncertainties’ of SMs are used to replace the original individual(s). In their work, the framework was presented for both single objective optimisation and multi-objective optimisation.

In principle, the present surrogate models used are implicitly or explicitly spatial meta-models, as their predictions involve exploiting some assumed spatial smoothness or continuity between the values of the objective function at a query point whose value is unknown and has to be predicted based on the known solutions in the search space. This makes objective meta-models naturally suited to continuous function optimisation. As such, the plethora of research studies that incorporate SMs to speed up evolutionary search on computationally expensive problems were made on problems involving continuous or real-valued variables. Particularly, if the input variables are real-valued, the simple Euclidean distance can be readily used. When the defined distance-measure correlates well with the fitness landscape, the meta-model is generally capable of approximating the search space. However, if the distance-measure just quantifies structural differences between solutions without consideration for their fitness values, then using the surrogate can mislead the search process since they may not approximate the search space well.

Based on the previous observation, Moraglio and Kattan [18] showed that SMs can be naturally generalised to encompass combinatorial spaces based in principle on any arbitrarily complex underlying solutions' representation by generalising their geometric interpretation from continuous to general metric spaces. Illustrative examples given are related to Radial Basis Function Networks (RBFNs) defined on the Hamming space associated with binary strings, which was used as a surrogate in GA system for optimising discrete combinatorial problems. These include the well-known NK-landscape problem.

In [10], the authors proposed a surrogate model called *GP-RBFN Surrogate* where GP has been used as an optimisation engine to optimise RBFN parameters in such a way as to improve its prediction accuracy. GP receives RBF functions as a primitive and tries to evolve a new RBF expression with different width parameters. The evolved RBFN has been used as a surrogate model to improve GA search. Experimental results on different MAX-SAT problems showed that GP-RBFN Surrogate improved the performance of GA search with a limited evaluation budget. The interesting remark about this work is that the authors implemented surrogate on a discrete problem and obtained good results.

Sun et al. [25], presented a two-layers surrogate-assisted Particle Swarm Optimisation (TLSAPSO) algorithm, in which a global and a number of local surrogate models are employed for fitness approximation. The model uses a global database of all particles found during the search in order to build the global surrogate model. Once the global model pinpoint a promising region, the system builds a local model using information from the neighbourhood around this point to enhance predictions made by the global model.

2.2. Surrogate Models for GP Representation

Kim et al. [12], argued that it is possible to extend surrogate model-based optimisation (SMBO) to more complicated representations which cannot be naturally mapped to vectors of features, if and only if, a suitable distance metric was defined. To this end, the authors showed that any surrogate model that can be written in terms of Euclidean distances between candidate solutions can be naturally generalised by replacing the Euclidean distance function with a generic metric appropriate to the target representation. In their experiments, RBFN has been used as a surrogate model on different benchmark problems for GA and GP. Different distance metrics have been used within the RBFN. The presented results for GP problems are not very different from those obtained with other methods when the distance metric was not suitable.

In [24], Schmidt and Lipson introduced a different model to approximate fitness landscape in GP using co-evolutionary approach. The proposed framework used three populations: 1) solutions to the original problem, evaluated using only fitness predictors; 2) fitness predictors of the problem; and 3) fitness trainers, whose exact fitness is used to train predictors. The fitness trainers are simply some selected pairs of individuals and their exact fitness measure which are used to evolve predictors. The predictors are used to approximate fitness of individuals in the first population. Empirical results with symbolic regression problems of this co-evolutionary approach showed a faster convergence rate than other GP system. The comparison was based on the needed number of point evaluations until the system finds solutions with error rate $< \epsilon$.

To the best of our knowledge, no previous work has successfully defined surrogate models on complex representations, such as the GP tree-like representation, other than real-valued vectors. When dealing with complex representation like the GP tree-like representation, there is no straightforward way to predict the fitness of a tree given its structural distance from another tree with known fitness, hence surrogate models cannot be easily used in GP search. Therefore, for search problems naturally based on structured representations, surrogate models can be used only after transforming the original representation to a real-valued vector form. In [19] and [12], an attempt was presented to apply RBFN surrogate model on GP, but it was concluded that the results reported were not significantly different from those obtained by a standard GP system.

3. Semantic Surrogate GP

The process of SSGP is broadly outlined in Figure 1. SSGP operates similarly to standard GP. The only difference residing in the fitness evaluation part. SSGP starts by randomly initialising a population using the ramped half-and-half method [13], and then uses standard sub-tree crossover and sub-tree mutation to explore the search space.

Assuming that one has a symbolic regression problem specified in a training set $T = \{(x_i, y_i)|i = 1, ..., n\}$ where x_i represents the i^{th} input vector and y_i is its associated output value. Consequently, the problem is to to obtain a function f that minimise some loss function (e.g., mean square error). Given that, the inputs are static in the sense that these do not change during the learning phase of the algorithm. Thus, the problem can be formulated in finding the function f that approximate as closely as possible the vector $t = (y_1, ..., y_n)$.

Fitness evaluation in GP can be viewed as a mapping from a syntactic space into a semantics space, and then as an additional mapping from a semantics space into a fitness value. Formally, this can be represented as follows, let S, V, and F be the syntactic, semantic, and fitness spaces, respectively. The function $Eval(tr, x_i) = v^{x_i}$ is the function that executes the code in tree tr using x_i as an input and produces its semantics (i.e., a vector of v^{x_i}'s outputs given the inputs), mapping $S \rightarrow V$. Furthermore, let $Fitness(v^{x_i}) = \hat{y}_i$ be the function that maps semantics into a fitness value. Thus, $Fitness(v^{x_i}) = \hat{y}_i$ maps $V \rightarrow F$, and can take the form of an arbitrary loss function, i.e., mean squared error or mean absolute error. The most expensive part of the mapping process (i.e., $S \rightarrow V \rightarrow F$) lies within the $Eval(tr, x_i)$ function. SSGP leverages on the $S \rightarrow V$ mapping by plugging a local surrogate model to substitute the $Eval(tr, x_i)$ function.

SSGP evaluates each tree within the population using a sub-training set. It then predicts each tree's outputs for the remaining fitness cases using a local surrogate model based on the simple K-Nearest Neighbour method. More formally, let a training set given to GP be the set $T = \{(x_i, y_i)|i = 1, ..., n\}$ and let a sub-set of T be $\mathbf{T_{sub}} = \{(x_i)|i = 1...z\}$ where $T_{sub} \subset T$ and $z << n$.

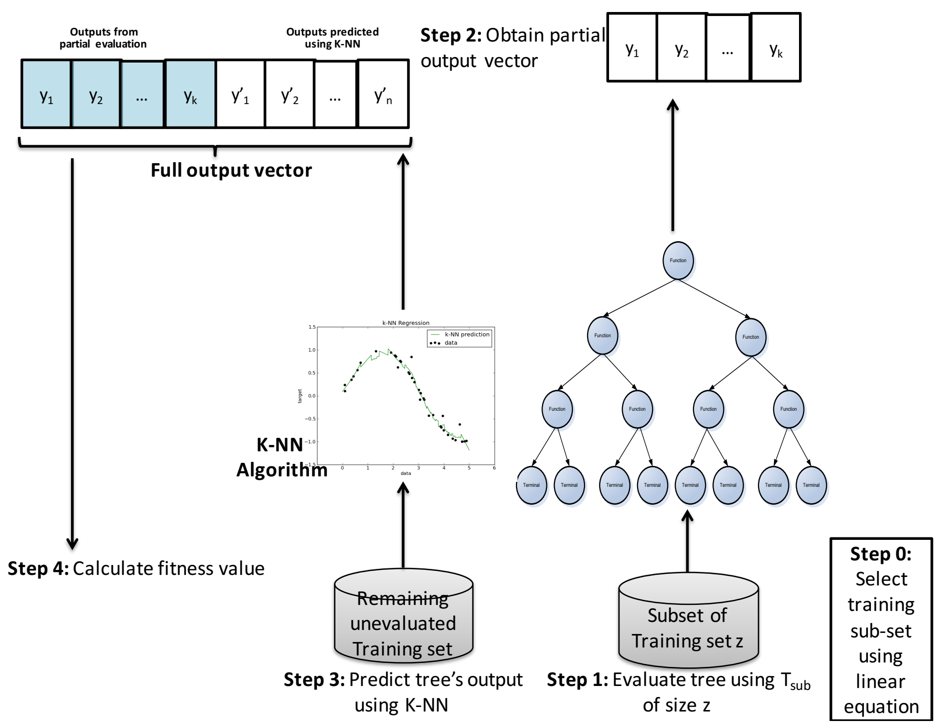

Outputs from partial evaluation
Outputs predicted using K-NN
Full output vector
Step 2: Obtain partial output vector
K-NN Algorithm
k-NN Regression
Step 4: Calculate fitness value
Remaining unevaluated Training set
Subset of Training set z
Step 3: Predict tree's output using K-NN
Step 1: Evaluate tree using T_{sub} of size z
Step 0: Select training sub-set using linear equation
Function
Terminal

Figure 1. Workflow outline of fitness evaluation in SSGP.

The size of the training sub-set $\mathbf{T_{sub}}$ is defined based on Algorithm 1. This simple algorithm allows the size of the training sub-set to increase or decrease based on the status of the search. For example, when an improvement in the best-so-far fitness is detected the algorithm assumes that the search is getting closer to a local optimum and thus reduces the computational cost (i.e., by reducing the size of the training sub-set). If the improvement stopped then it will gradually increase the size of the training sub-set allowing the fitness measure to be more deterministic. Finally, if no improvement was detect for a fixed number of generations then it will decide to save computational cost and will start reducing the size of the training sub-set. The size of the training sub-set will be capped between $99\% - 5\%$.

Algorithm 1 starts from an initial size for the training sub-set at 99% of training set original size. Then, it simply reduces the size of the training set every time the *best-so-far* fitness improves. If no further improvements was detected then it decides to gradually increase the size of the training sub-set (at steps of 0.1% every generation). However, if no improvement in the best-so-far fitness was detected for 20 continuous generations then the algorithm will assume that the search is trapped in local optimum and will reduces the training sub-set gradually. The max step size of reducing training sub-set defined as 5% and the value of reduction will be scaled based on the number of generation. The idea of dynamically select a sub-set of the training set has been addressed in the literature before (e.g., see Dynamic Subset Selection and Historical Subset Selection in [5]).

Algorithm 1: Size of training sub-set.

```
1  var reduce_step = $0.05$;
2  var increase_step = $0.001$;
3  var platu = $20$;
4  sub_set = $0.99$;
5  var scale = $\frac{index\_current\_generation}{max\_number\_of\_generations}$;
6  if IsImproved(Best_so_far_fitness) is true then
7      sub_set = sub_set - (reduce_step $\times$ scale)
8  end
9  if IsImproved(Best_so_far_fitness) == false then
10     sub_set = sub_set + increase_step
11 end
12 if IsImproved(Best_so_far_fitness) == false AND No Improvement for a fixed number
   of generations then
13     sub_set = sub_set - (reduce_step $\times$ scale)
14 end
15 return Training sub_set size = sub_set $\times$ TrainingSet size;
```

Note that the system evaluates the trees using the sub-training set (which has dynamic size as explained above) and uses KNN algorithm to predict the trees' outputs on the remaining un-evaluated training examples (more details in the next section).

3.1. K-Nearest Neighbour Surrogate

SSGP uses a local surrogate model to predict tree-output based on the K-Nearest-Neighbour (K-NN) method [1]. A training sample $x_i \in \mathbb{R}^d$ is represented as a d-dimensional vector of real values. For each tree in the population, the system calculates the outputs $t = \{y_0, ..., y_z\}$ of the sub-training set $\mathbf{T_{sub}}$. The remaining unevaluated training samples which are in set **Diff** = $T - \mathbf{T_{sub}}$ are mapped to tree-outputs using the K-NN. A predicted output is simply the average of outputs for the K nearest input vectors in $\mathbf{T_{sub}}$.

4. Empirical Evaluation

The main aim of this paper is to introduce the notion of surrogate to the GP semantic space and reduce time of the training process. However, we want to achieve this without significantly compromise the quality of evolved solutions. The contribution of the proposed SSGP is in its training time which as will be shown by the experiments, SSGP several order of magnitude faster than other GP systems. In this paper, we did not focused on the execution time of the testing phase (i.e., execution of the evolved program on the testing set).

In terms of performance, we show that SSGP achieves solutions that are not far from those obtained by other GP systems. More importantly, the quality of SSGP's solutions are similar to the other GP systems included in the comparison (statistically tested). Hence, the application of SSGP makes an excellent added value for practitioners who are interested in evolving acceptable solutions (similar to those obtained by canonical GP systems) in shorter CPU time.

The experiments included 17 test problems divided into three categories. Namely, 9 classifications problems from the UCI repository [23], 3 time-series forecasting problems, and 5 symbolic regression problems. The experiments will focus on two main aspects of the comparison; A) Execution time (of training phase), and B) Performance. In this experiments, we used desktop PC with Intel core i7 processors, 2.40 GHz 8GB RAM, running 64-bit Windows 8 OS. The execution time was measured at code level using CTime library in c++. As will be shown in the next sub-sections, SSGP has lower training time than all competitors, in most cases, at significant margins. While, all algorithms have close results in terms of performance.

4.1. Experimental Settings

The aim of the reported experiments is to investigate the performance of SSGP under different circumstances. In our experiments, we compared SSGP against: 1) standard GP (SGP), 2) steady state GP (steadyGP) [28], 3) GP with random training sampling technique (RT-GP) [7], and 4) GP with dynamic random sub-set selection using the technique described in Algorithm 1 (DT-GP). Standard GP was selected in the comparison because it can be considered as the baseline system. Steady state GP was selected because it has quick convergence rates which makes it a good competitor to SSGP in finding good solutions within a limited number of evaluations. GP with a random sampling technique was selected in the comparison because it also evaluates trees using sub-training set (based on a randomly selected sub-set of the complete training sample). Finally, GP with dynamic random sub-set selection using the technique described in Algorithm 1 is selected to test whether KNN prediction will add value to the search or not.

All GP systems included in the experiments received exactly the same settings as presented in table 1. For RT-GP, we set the size of the sub-set at 50% of the of original training set, randomly selected and change in every generation. For SSGP and DT-GP, we set the initial size of the training sub-set at 99% of the original training set.

4.2. Classification problems

For classification problems, we used 9 problems, presented in table 2, to test SSGP performance. In this paper, we used GP trees' output as a range index to perform classification. For example, if the number of classes is 3 (like in the *Balance scale* problem), then GP trees are expected to produce outputs < 0 for instances from class 1, outputs from $[0, 1]$ for instances from class 2, and outputs > 1 from instances from class 3. The fitness measure is simply the total number of misclassified training instances (i.e., a minimisation problem). All GP systems are equipped with the function set presented in table 3. As presented in the table, we added a set of statistical measures that are used to extract new features from the

Table 1. All GP systems settings used in the experiments.

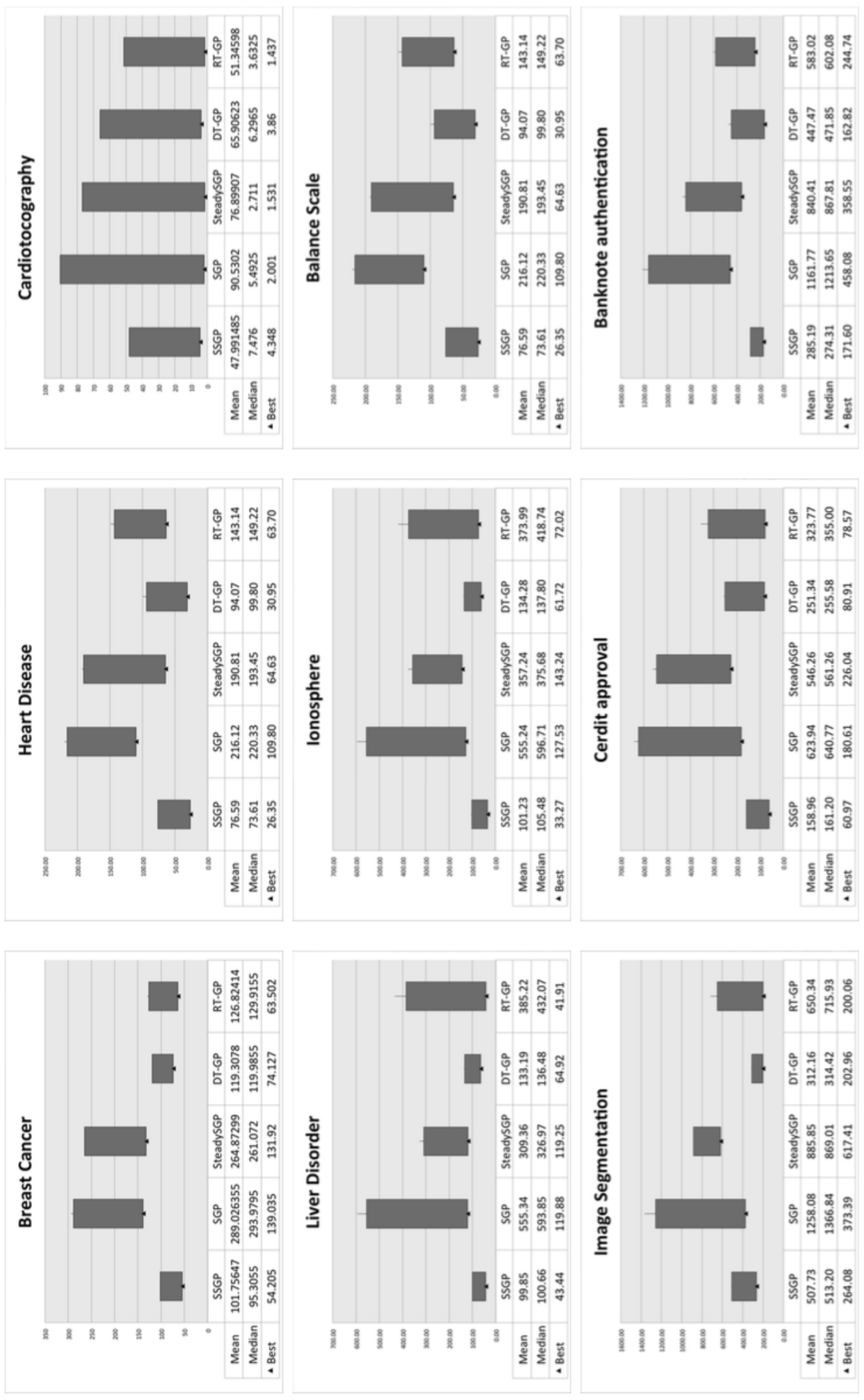
Parameter	*Value*
Sub-tree Mutation	30%
Sub-tree Crossover	70%
Tournament size	2
Population Size	200
Generations	200
Initialisation method	ramped half-and-half
SSGP special parameters	
K (in K-NN)	10
Initial training sub-set size	99% of original training set
DT-GP special parameters	
Initial training sub-set size	99% of original training set
RT-GP special parameters	
training sub-set size	50% of original training set

attribute set of each problem. We found this, generally, enhance the performance of GP solutions. We divided the data into three disjoint sets as follows; 25% training set, 25% validation set, and 50% testing set. The best evolved solution in each generation was further evaluated using the validation set and the best solution that has the best performance on the validation set across the whole run was selected as the final evolved solution [1].

As mentioned previously, SSGP evaluates fitness function partially, using the training sub-set, and then use KNN to predict the trees' output for the remaining unevaluated cases. In this problem, KNN employed a majority voting scheme to predict the output of trees when evaluating any unevaluated instance.

Table 2. Classification problems used in the experiments from UCI repository [23].

Problem	Number of Instances	Number of Attributes	Number of Classes
Breast Cancer	286	9	2
Heart Disease	303	75	4
Ionosphere	351	34	2
Credit Approval	690	15	2
Banknote authentication	1372	5	2
Image Segmentation	2310	19	7
Cardiotocography	2126	23	3
Balance Scale	625	4	3
Liver Disorders	345	7	2



Breast Cancer

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	101.75647	289.026355	264.87299	119.3078	126.82414
Median	95.3055	293.9795	261.072	119.9855	129.9155
▲ Best	54.205	139.035	131.92	74.127	63.502

Heart Disease

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	76.59	216.12	190.81	94.07	143.14
Median	73.61	220.33	193.45	99.80	149.22
▲ Best	26.35	109.80	64.63	30.95	63.70

Cardiotocography

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	47.991485	90.5302	76.89907	65.90623	51.34598
Median	7.476	5.4925	2.711	6.2965	3.6325
▲ Best	4.348	2.001	1.531	3.86	1.437

Liver Disorder

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	99.85	555.34	309.36	133.19	385.22
Median	100.66	593.85	326.97	136.48	432.07
▲ Best	43.44	119.88	119.25	64.92	41.91

Ionosphere

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	101.23	555.24	357.24	134.28	373.99
Median	105.48	596.71	375.68	137.80	418.74
▲ Best	33.27	127.53	143.24	61.72	72.02

Balance Scale

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	76.59	216.12	190.81	94.07	143.14
Median	73.61	220.33	193.45	99.80	149.22
▲ Best	26.35	109.80	64.63	30.95	63.70

Image Segmentation

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	507.73	1258.08	885.85	312.16	650.34
Median	513.20	1366.84	869.01	314.42	715.93
▲ Best	264.08	373.39	617.41	202.96	200.06

Cerdit approval

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	158.96	623.94	546.26	251.34	323.77
Median	161.20	640.77	561.26	255.58	355.00
▲ Best	60.97	180.61	226.04	80.91	78.57

Banknote authentication

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	285.19	1161.77	840.41	447.47	583.02
Median	274.31	1213.65	867.81	471.85	602.08
▲ Best	171.60	458.08	358.55	162.82	244.74

Figure 2. Summary of CPU execution time of 9000 independent GP runs. Results are collected from (200) independent runs for each GP system. Numbers represent the GP training time in seconds.

Table 3. GP systems function-primitive sets for classification problems

Function set	+, -, /, *
statistical features	Mean, Median, StD, AverageDev, Entropy
Constants	10 constants from $[1, 10]$

*StD = Standard Deviation, AverageDev = Average Deviation

Figures 2, summarise the CPU execution time (in seconds) of 9000 GP runs (in the training process). Each GP system tried to evolve classifiers for the given problems through 200 independent runs. We reported the mean, median, and best CPU execution time across the whole runs. As can be clearly seen, SSGP has the lowest execution time in terms of mean, median and best in 8 out of 9 cases. This is remarkable results given the margin of differences. The only two cases where SSGP was slower are the *Cardiotocography* and the *Image Segmentation* problems. To further verify the significance of these results, we performed non-parametric Friedman statistical test to rank the five algorithms tested. As it can observed from table 4, the SSGP execution time was ranked first in 7 out of the 9 test problems; in addition, Holm's post-hoc test [6] showed that in all 9 problems the first ranking were statistically significant at 5% level.

Now, if we turn our attention to the classification errors. Table 5 shows the classification errors (on the testing set) presented as percentages for all GP systems in all test problems. As shown in the table, there is no single algorithm dominates all other in all test problems (or at least in most of them). In addition, the performance differences between all five algorithms are marginal. To further verify this, table 6 presents the Friedman statistical test. The lowest rank algorithm was interchanging between SGP and SteadyGP. The P-Value vary less and higher than 5% significance level. In the cases where P-Value is statistically significant the performance difference is very small, as shown in table 5.

Table 4. Friedman statistical test to compare execution times. **Bold** numbers are the lowest ranks and last column shows P-value of all system against the system with the lowest rank. We used the notation < 0.05 to show that the GP system with the lowest rank is statistically significant than all other GP systems at 5% significance level.

Classification Problem	**SSGP**	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**	**P-Value**
Breast Cancer	**1.34**	4.66	4.34	2.15	2.51	< 0.05
Cardiotocography	4.01	3.54	**1.66**	3.74	2.06	< 0.05
Ionosphere	**1.07**	4.86	3.46	1.96	3.65	< 0.05
Credit Approval	**1.08**	4.64	4.35	2.10	2.82	< 0.05
Banknote authentication	**1.10**	4.89	4.06	1.98	2.97	< 0.05
Image Segmentation	2.22	4.75	4.09	**1.19**	2.75	< 0.05
Heart Disease	**1.26**	4.68	4.21	1.82	3.03	< 0.05
Liver Disorders	**1.11**	4.90	3.26	1.98	3.77	< 0.05
Balance Scale	**1.10**	4.85	3.60	2.39	3.06	< 0.05

Table 5. Summary of classification error of 9000 independent GP runs. Each system was tested through 200 independent runs. Results shows the classification error rates in percentages (on the testing set). **Bold** numbers are the lowest.

	SSGP	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**
			Breast Cancer		
Mean	**0.03**	0.04	**0.03**	**0.03**	0.05
Median	**0.03**	0.04	**0.03**	**0.03**	0.04
Best	**0.01**	0.02	0.02	0.02	0.02
			Cardiotocography		
Mean	**0.13**	**0.13**	0.21	**0.13**	**0.13**
Median	**0.13**	**0.13**	0.22	**0.13**	**0.13**
Best	**0.02**	0.05	**0.02**	**0.02**	0.05
			Ionosphere		
Mean	0.12	0.16	**0.11**	0.12	0.17
Median	**0.11**	0.16	**0.11**	**0.11**	0.17
Best	0.06	0.08	**0.05**	0.05	0.07
			Credit Approval		
Mean	0.21	0.17	0.20	0.21	**0.17**
Median	0.18	0.16	0.16	0.17	**0.15**
Best	0.13	0.13	0.13	0.13	**0.12**
			Banknote Authentication		
Mean	0.03	**0.02**	0.03	0.03	0.03
Median	0.03	**0.02**	0.03	0.03	**0.02**
Best	0.003	**0.00**	0.01	**0.00**	**0.00**
			Image Segmentation		
Mean	0.60	**0.51**	0.60	0.60	0.52
Median	0.61	**0.48**	0.60	0.60	0.49
Best	0.45	**0.36**	0.55	0.45	0.37
			Heart Disease		
Mean	0.44	**0.43**	0.63	0.44	**0.43**
Median	0.46	**0.45**	0.69	0.46	0.47
Best	0.30	0.18	**0.15**	0.26	0.20
			Liver Disorder		
Mean	0.36	**0.32**	0.37	0.36	0.33
Median	0.37	**0.31**	0.37	0.37	0.32
Best	0.28	**0.26**	0.27	**0.26**	0.27
			Balance Sclae		
Mean	0.44	**0.43**	0.64	0.44	**0.43**
Median	**0.46**	0.45	0.69	**0.46**	0.47
Best	0.30	0.19	**0.15**	0.26	0.20

***Bold** numbers are the lowest

4.3. Symbolic regression problems

For the symbolic regression problems, we included five test problems (described in table 8). All GP systems were guided to minimise the mean of absolute errors. Furthermore, all GP systems received the same functions-primitive sets in table 7. To guide the evolutionary process, a training set of 200 samples was randomly generated from the closed interval $[-5, 5]$. In addition, a validation and testing sets of 100 and 500 samples, respectively, were randomly generated from same set interval. In this problem, we used weighted KNN to predict the trees outputs when evaluating unseen instances from the training set. The KNN's prediction was calculated as follows:

$$\hat{y} = \sum_{i=1}^{N} \frac{1}{w_i} \times TreeOutput(x_i)$$

where w_i is the euclidean distance between unseen point and the i^{th} nearest neighbour.

Figure 3 summarise the CPU execution time of 5000 GP runs (for the training process). Each GP system tried to evolve approximation for each target function in 200 independent runs. The figures report the mean, median, and best solution across the whole 200 runs. As can be seen from the figures, SSGP achieved the lowest mean in all runs and the best median in 3 out of 5 problems. Similar to the classification problem, we verified the significance of the results using a non-parametric Friedman statistical tests (presented in table 9). Clearly, SSGP achieved the lowest rank in 4 out of 5 test problems however the significance level was more than 10%.

Now, if we compare the performance of the five GP systems in terms of approximation error (presented in table 10), similar to the classification problem, all GP systems evolved nearly similar solutions. The differences are marginal in all test problems. However, the differences of performance in problems $F4$ and $F5$ is larger than the other three problems. Furthermore, in table 11, we presented the results of the Friedman statistical test to compare the quality of performance. As can be seen, there is no single algorithm achieved the lowest rank consistently across all problems. Standard GP was statistically better than other algorithms in 2 out of 5 problems (i.e., $F4$ and $F5$).

Generally, SSGP may show poor performance in regression problems than other GP systems. One reason may be the complexity of the landscape in some regression problems makes the KNN predictions misleading and thus SSGP may produce inferior solutions than other systems.

Table 6. Friedman statistical test to compare performance. the last column shows P-value of all system against the system with the lowest rank.

Rank	Algorithm	P-Value
Breast Cancer - Lowest Rank: SteadyGP		
4	RT-GP	2.38E-21
3	SGP	1.20E-14
2	SSGP	0.01
1	DT-GP	0.88
Cardiotocography - Lowest Rank: SGP		
4	SteadyGP	1.25E-43
3	SSGP	0.051
2	DT-GP	0.31
1	RT-GP	0.41
Ionosphere - Lowest Rank: SteadyGP		
4	RT-GP	3.48E-34
3	SGP	3.86E-31
2	DT-GP	0.07
1	SSGP	0.10
Credit Approval - Lowest Rank: SGP		
4	DT-GP	7.71E-10
3	SSGP	5.93E-9
2	SteadyGP	3.74E-4
1	RT-GP	0.89
Banknote - Lowest Rank: SGP		
4	SSGP	1.17E-11
3	RT-GP	2.76E-7
2	DT-GP	4.20E-7
1	SteadyGP	4.20E-6
Image Segmentation - Lowest Rank: SGP		
4	SteadyGP	8.326E-70
3	SSGP	1.17E-16
2	DT-GP	2.60E-16
1	RT-GP	0.22
Heart Disease - Lowest Rank: SGP		
4	SteadyGP	2.43E-43
3	DT-GP	0.009
2	RT-GP	0.01
1	SSGP	0.03
Liver Disorder - Lowest Rank: SGP		
4	SteadyGP	1.53E-14
3	SSGP	7.91E-13
2	DT-GP	2.77E-12
1	RT-GP	0.04
Liver Disorder - Lowest Rank: SteadyGP		
4	SSGP	6.00E-32
3	RT-GP	2.61E-20
2	SGP	1.17E-16
1	DT-GP	3.43E-15

Table 7. GP systems function-primitive sets for symbolic regression problems.

Function set	+, -, /, *, pow, square root
Constants	10 constants from $[1, 10]$

Table 8. Symbolic regression problems

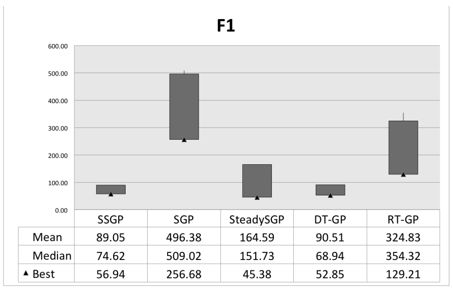
$$F_1 : \quad f(x_1, x_2) = \frac{e^{-(x_1-1)^2}}{1.2+(x_2-2.5)^2} \qquad F_2 : \quad f(x_1, x_2, x_3, x_4, x_5) = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$$

$$F_3 : \quad f(x_1, x_2) = 6sin(x_1)cos(x_2) \qquad F_4 : \quad f(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3x_4x_5$$

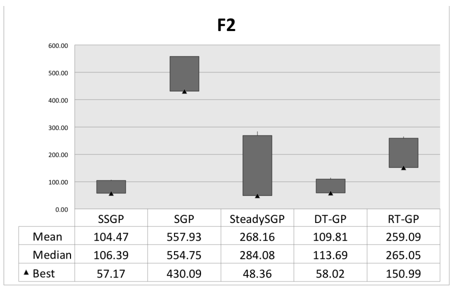
$$F_5 : f(x_1, x_2, x_3, x_4, x_5) = 32 - 3\frac{tan(x_1)}{tan(x_2)}\frac{tan(x_3)}{tan(x_4)}$$

Table 9. Friedman statistical test to compare execution times. **Bold** numbers are the lowest ranks and last column shows P-value of all system against the system with the lowest rank. We used the notation $<$ and $>$ to show whether the GP system with the lowest rank is statistically significant than all other GP systems at 5% or 10% significance level.

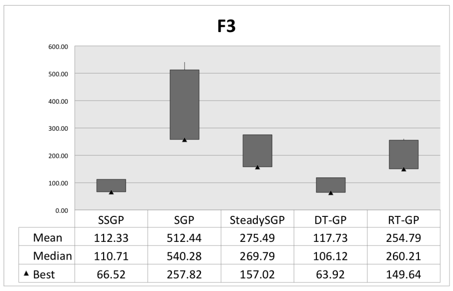
	SSGP	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**	**P-Value**
F1	2.02	4.775	2.66	**1.86**	3.685	> 0.1
F2	**1.525**	5.00	3.47	1.62	3.38	> 0.1
F3	**1.535**	4.85	3.70	1.60	3.31	> 0.1
F4	**1.52**	4.85	3.91	1.69	3.02	> 0.1
F5	**1.495**	4.85	3.90	1.60	3.14	> 0.1



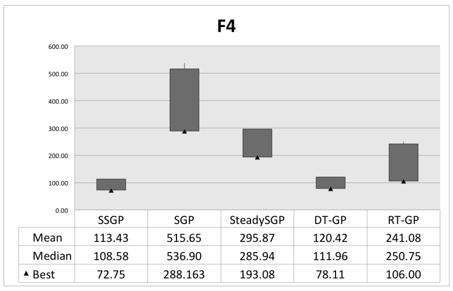
	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	89.05	496.38	164.59	90.51	324.83
Median	74.62	509.02	151.73	68.94	354.32
▲ Best	56.94	256.68	45.38	52.85	129.21



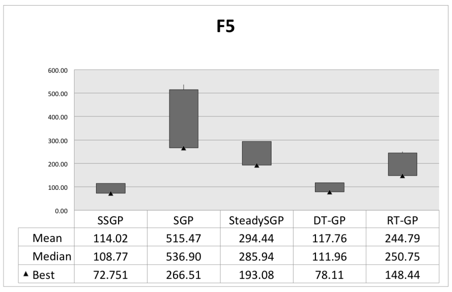
	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	104.47	557.93	268.16	109.81	259.09
Median	106.39	554.75	284.08	113.69	265.05
▲ Best	57.17	430.09	48.36	58.02	150.99



	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	112.33	512.44	275.49	117.73	254.79
Median	110.71	540.28	269.79	106.12	260.21
▲ Best	66.52	257.82	157.02	63.92	149.64



	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	113.43	515.65	295.87	120.42	241.08
Median	108.58	536.90	285.94	111.96	250.75
▲ Best	72.75	288.163	193.08	78.11	106.00



	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	114.02	515.47	294.44	117.76	244.79
Median	108.77	536.90	285.94	111.96	250.75
▲ Best	72.751	266.51	193.08	78.11	148.44

Figure 3. Summary of CPU execution time of 750 independent GP runs. Results are collected from (30) independent runs for each GP system. Numbers represent the execution time in seconds. This time is for the GP training phase.

Table 10. Summary of approximation errors of 5000 independent GP runs. Each system was tested through 200 independent runs. Results shows the mean of absolute errors. **Bold** numbers are the lowest.

	SSGP	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**
F1					
Mean	0.07	**0.06**	4.77	**0.06**	**0.06**
Median	**0.05**	**0.05**	**0.05**	**0.05**	**0.05**
Best	0.02	**0.01**	**0.01**	0.02	0.02
F2					
Mean	0.14	0.08	**0.07**	**0.07**	0.09
Median	0.06	0.08	**0.05**	0.06	0.09
Best	0.03	**0.02**	0.03	**0.02**	0.04
F3					
Mean	**2.31**	2.62	2.39	2.34	2.57
Median	**2.28**	2.36	2.30	2.29	2.38
Best	2.03	1.70	**1.46**	1.78	1.90
F4					
Mean	76.08	44.14	69.78	76.19	**66.61**
Median	74.54	**1.86**	63.98	74.14	103.88
Best	34.46	**2.741E-05**	0.33	0.44	2.75E-05
F5					
Mean	87.43	**58.13**	82.22	88.02	77.94
Median	74.54	**2.53**	64.62	74.29	103.88
Best	34.45	**2.74E-05**	26.50	29.57	$2.75E-05$

Table 11. Friedman statistical test to compare performance. the last column shows P-value of all system against the system with the lowest rank.

Rank	Algorithm	P-Value
F1 - Lowest Rank: DT-GP		
4	SteadyGP	0.009
3	SSGP	0.282
2	RT-GP	0.319
1	SGP	0.527
F2 - Lowest Rank: SteadyGP		
4	RT-GP	8.91E-35
3	SGP	1.11E-21
2	DT-GP	0.13
1	SSGP	0.164
F3 - Lowest Rank: DT-GP		
4	RT-GP	5.01E-16
3	SGP	2.22E-14
2	SteadyGP	0.29
1	SSGP	0.83
F4 - Lowest Rank: SGP		
4	DT-GP	8.46E-12
3	RT-GP	3.68E-9
2	SSGP	5.39E-9
1	SteadyGP	8.77E-7
F5 - Lowest Rank: SGP		
4	SSGP	5.16E-10
3	DT-GP	4.90E-9
2	SteadySGP	8.24E-6
1	RT-GP	1.96E-5

4.4. *Time-series forecasting problems*

For the time-series forecasting problems, we used data from *Google Trends* service [8]. Google Trends is a free service provided by Google offering data about the search terms that people entered into Google search engine. The service provides free downloadable historical time-series data about any keyword. It, also, offers the flexibility to restrict the search by country. One of the uses of Google Trends is for E-Marketing managers to monitor the internet to see how often people type certain keywords related to their products at different times over the year. Using this information, E-Marketing managers can decide the best time to release their marketing campaigns so their advertisements coincide with people's searchers and eventually achieve higher hit rates.

For the purpose of our experiments, we imported time-series data about people's searches for the keywords; *Jobs, Holidays*, and *Cinema* and we restricted the search to get data from *UK, USA* and *USA*, respectively. All imported data from Google Trends represent the weekly frequencies of searches of the keywords mentioned above in Google since January 2004 until December 2014. There are 571 data points.

All GP systems received the 5 points (under a sliding window of size 5 and moving steps of size 1) from x_i to x_{i+5} and asked to predict point x_{i+6}. All GP systems received primitive-function sets presented in table 3. Time-series data were divided in three disjoint sets (25% training, 25% validation, and 50% testing). The evolutionary process guided to minimise the mean of absolute errors. The weighted KNN was used as described in the previous sub-section.

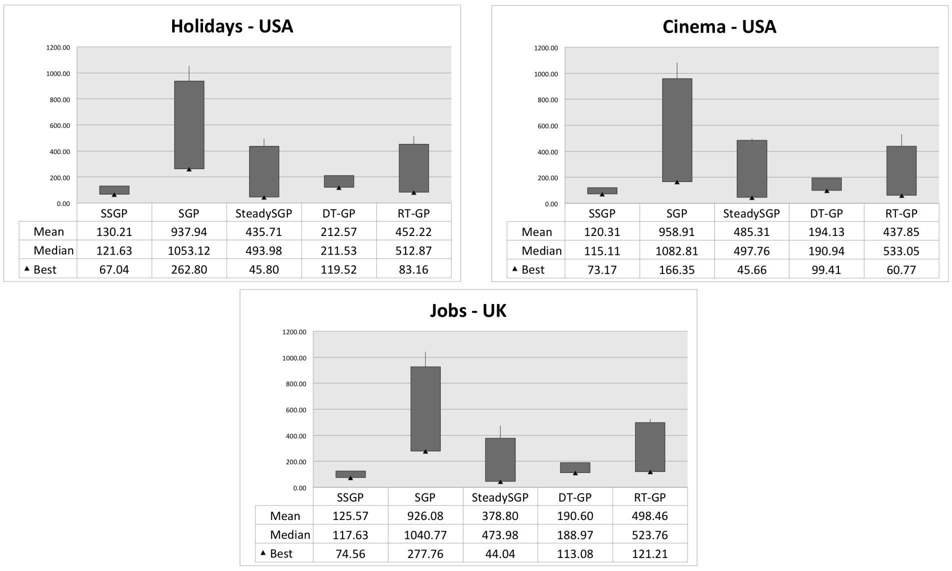
Following the same style as the previous sub-section of presenting the results. First we summarise the CPU execution times of 3000 GP runs (for the training process). Figure 4 shows that SSGP has the lowest execution time in terms of mean, median in all problems. Friedman statistical test shows that SSGP is statistically significant than the other four GP systems at 5% significance level.

For the forecast errors, presented in table 13, clearly there is no single algorithm dominate all other algorithms in any problem. Moreover, errors are very close. To further verify this, table 14 presents Friedman statistical test. SGP achieved the lowest rank in all test problems. However, the p-value is larger than 5% statistical significance level in most of the cases.

From the previous results, it can be concluded that SSGP is faster than other GP systems and it's results is not too far from other competitive systems. This confirms that SSGP has the added value of evolving good solutions in lower time than other existing techniques.

4.5. *Discussion*

4.5.1. KNN Added Value SSGP achieves it's fast CPU execution time due to two main components. First we use simple linear equation to control the size of the training sub-set



Holidays - USA

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	130.21	937.94	435.71	212.57	452.22
Median	121.63	1053.12	493.98	211.53	512.87
▲ Best	67.04	262.80	45.80	119.52	83.16

Cinema - USA

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	120.31	958.91	485.31	194.13	437.85
Median	115.11	1082.81	497.76	190.94	533.05
▲ Best	73.17	166.35	45.66	99.41	60.77

Jobs - UK

	SSGP	SGP	SteadySGP	DT-GP	RT-GP
Mean	125.57	926.08	378.80	190.60	498.46
Median	117.63	1040.77	473.98	188.97	523.76
▲ Best	74.56	277.76	44.04	113.08	121.21

Figure 4. Summary of CPU execution time of 3000 independent GP runs. Results are collected from (200) independent runs for each GP system. Numbers represent the execution time in seconds.This time is for the GP training phase.

Table 12. Friedman statistical test to compare execution times. **Bold** numbers are the lowest ranks and last column shows P-value of all system against the system with the lowest rank. We used the notation $<$ and $>$ to show whether the GP system with the lowest rank is statistically significant than all other GP systems at 5% or 10% significance level.

	SSGP	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**	**P-Value**
Holidays - USA	**1.195**	4.91	3.23	2.28	3.38	< 0.05
Cinema - USA	**1.195**	4.85	3.53	2.18	3.24	< 0.05
Jobs- UK	**1.305**	4.91	2.85	2.3	3.63	< 0.05

Table 13. Summary of forecast errors of 3000 independent GP runs. Each system was tested through 200 independent runs. Results shows the mean of absolute errors. **Bold** numbers are the lowest.

	SSGP	**SGP**	**SteadyGP**	**DT-GP**	**RT-GP**
Holidays - USA					
Mean	5.04	5.55	5.28	5.10	**5.01**
Median	**4.93**	**4.93**	**4.93**	**4.93**	**4.93**
Best	4.41	**3.96**	4.15	4.46	4.32
Cinema - USA					
Mean	6.45	**6.30**	6.39	6.40	10.09
Median	6.49	**5.98**	6.36	6.35	6.23
Best	6.01	**5.53**	**5.53**	5.85	5.70
Jobs - UK					
Mean	**3.43**	3.58	3.45	3.45	3.44
Median	**3.46**	**3.46**	**3.46**	**3.46**	**3.46**
Best	3.25	**3.22**	3.24	3.25	**3.22**

Table 14. Friedman statistical test to compare performance. the last column shows P-value of all system against the system with the lowest rank.

Rank	Algorithm	P-Value
Holidays - Lowest Rank: SGP		
4	DT-GP	0.24
3	SSGP	0.429
2	SGP	0.86
1	RT-GP	0.98
Jobs - Lowest Rank: SGP		
4	DT-GP	1.11E-6
3	SteadyGP	7.72E-5
2	SSGP	0.001
1	RT-GP	0.17
Holidays - Lowest Rank: SGP		
4	SSGP	1.51E-35
3	DT-GP	8.41E-19
2	SteadyGP	2.69E-17
1	RT-GP	8.52E-10

(see Algorithm 1 for more details). This allows SSGP to dynamically change the training sub-set size based on the status of the search. However, this component by itself is not enough to fully accelerate the evolutionary process. As we have seen in the previous three sub-sections, SSGP is faster than DT-GP in most test cases. Note that DT-GP is similar to SSGP with the only exception that it does not use KNN to predict trees' outputs. DT-GP infers the fitness directly from the training sub-set (similar to RT-GP). The second component that strongly supports the first component is KNN surrogate which is used to predict the full trees' semantic representation based on the evaluated sub-set. Naturally, KNN predictions are not totally error free. However, we believe that the errors produced by the KNN help to smooth a rugged search space in most cases. As illustrated in figure 6, we compared the size of the training sub-set between DT-GP and SSGP generation-by-generation (data in the figures are averaged over 200 independent runs for each system). In the top two charts in figure 6 (i.e., Breast Cancer and Cardiotocography), the size of the sub-set drops quickly in SSGP. This is because, as described in Algorithm 1, when there is no improvement in the fitness for 20 consecutive generations the size of the sub-set will drop gradually at different steps within 5% from its initial size. Note that SSGP were faster than DT-GP in these two test problems (see table 2). Here, the KNN has helped to smooth the search space where solutions in SSGP reached a local optimum quickly. Interestingly, the quality of solutions located in the local optimum (in the SSGP search space) are not too far from the original search space where GP trees are evaluated directly without KNN support (which indicate that the KNN predictions are not too noisy). In the bottom chart in figure 6 (i.e., entitled Image segmentation), the size of the sub-set drops quicker in DT-GP. This is an indication that the SSGP requires longer time to reach the local optimum of the search space in this particular problem.

Overall, we believe that the differences in SSGP CPU execution times in different test problems is largely attributed to the ability of the KNN to smooth the search space (see [22]). Figure 5 shows the predictions of KNN for fitness cases versus the true trees' outputs (for Holidays time-series forecasting problem). The data on the scatter plot take a linear shape which indicate that KNN predictions and not largely deviated from the true outputs. The coefficient correlation between KNN predictions and true trees' outputs is around 0.60, in this particular problem. Naturally, KNN predictions' accuracy increases as the size of the $\mathbf{T_{sub}}$ increases.

To understand different aspects of SSGP we also compared the average trees sizes generation-by-generation in all GP systems (averaged from 200 runs for each GP system). Illustrated in figure 7 the average trees size in three classification problems. Namely, *Breast Cancer problem*, *Credit Approval problem* and, *Image Segmentation problem* (average tree sizes of other test problems are not shown here, but the GP systems' behaviour is almost consistent in all test problems). Generally, SSGP and DT-GP produce the smallest growth in tree sizes among all GP systems in the comparison. Both SGP and RT-GP showing consistent behaviour and their average tree sizes are the highest in all problems. Finally, for steadyGP, its trees growth fall in between the two groups (i.e, SSGP & DT-GP and SGP & RT-GP). We artificially capped trees to be no more than 150 nodes prevent programs bloating.

Moreover, in the same line of understanding different aspects of SSGP, we provided a sensitivity analysis to fine the best value of *k* in the KNN algorithm. Figure 8 shows

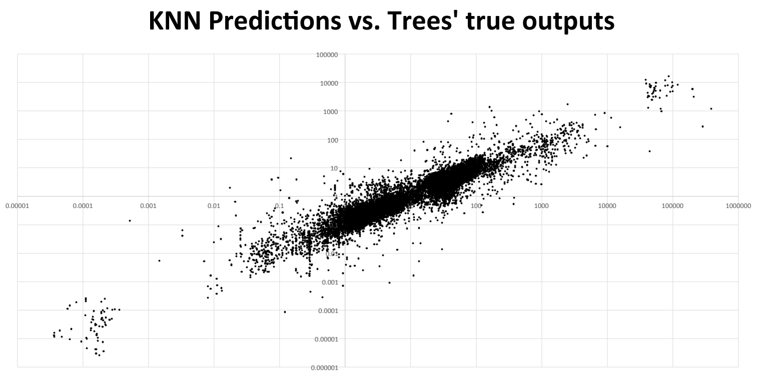


Figure 5. Scatter plot shows KNN predictions of fitness cases vs. trees' true outputs for Holidays time-series problem. The x-axis is the true outputs and the y-axis is the predicted outputs.

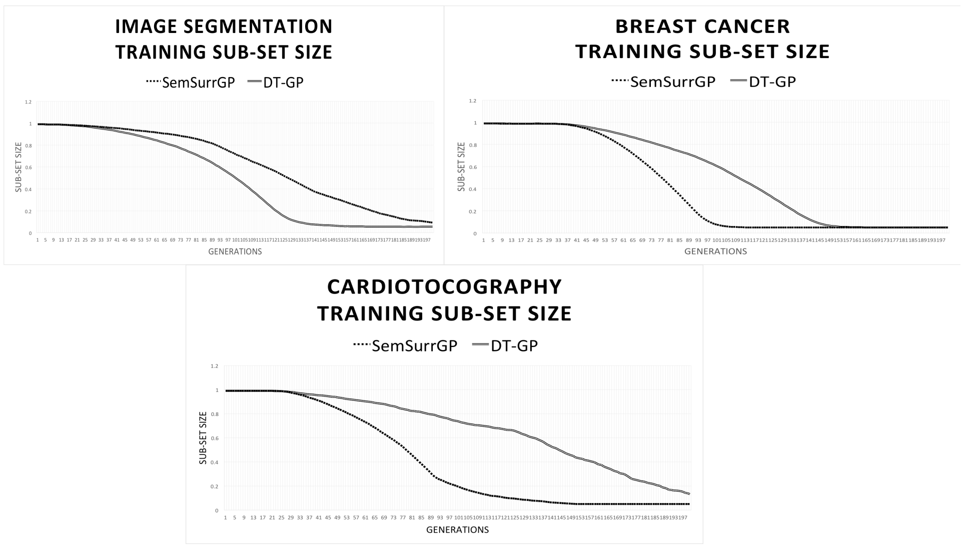


Figure 6. A comparison of training sub-set sizes (generation-by-generation) between DT-GP and SSGP. The figures shows three classification. Top left the *Breast Cancer problem*, top right the *Cardiotocography problem* and at the bottom the *Image Segmentation problem*.

SSGP performance in the Credit Approval classification problem under different values of k. Clearly the system prefers larger values of k to support the KNN predictions. We tested the system with different values of k (from 2 to 10).

In the current realisation, the implementation of the KNN is based on Euclidean distances which may suffer to produce good approximations in high-dimensional spaces. To

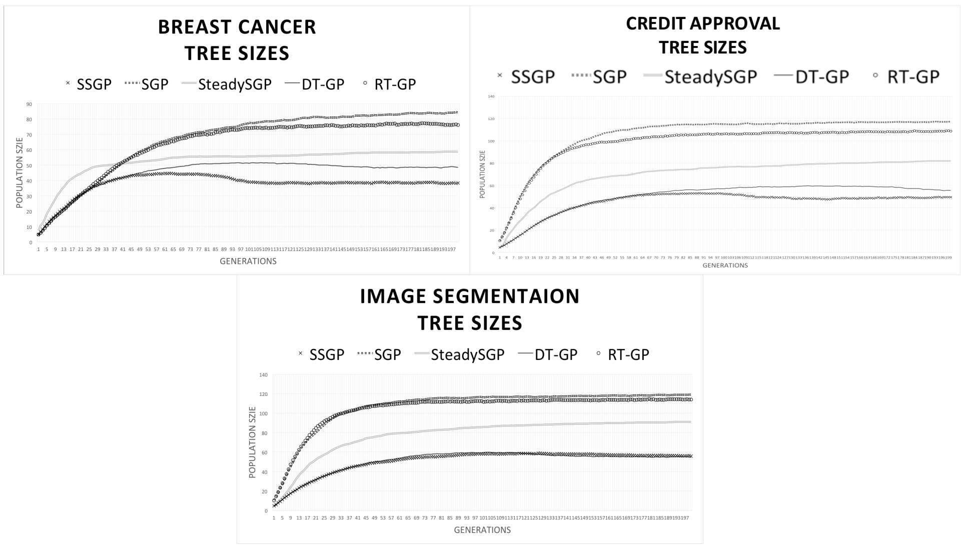


Figure 7. A comparison of average population size (generation-by-generation) between DT-GP and SSGP. The figures shows three classification. Top left the *Breast Cancer problem*, top right fitness the *Credit Approval problem* and at the bottom the *Image Segmentation problem*.

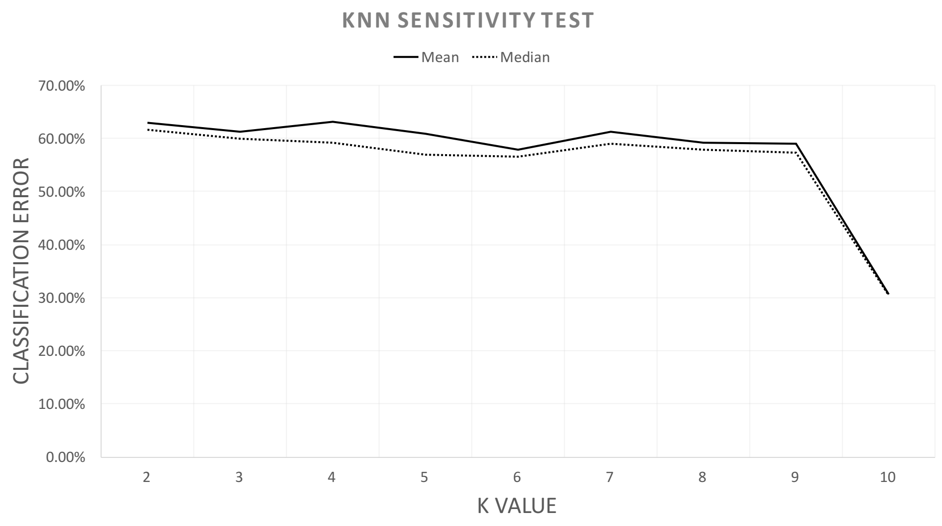


Figure 8. KNN sensitivity test to show the performance with different values of *K*. For each K value we tested the system 20 different times and reported the mean and median. The test problem was in Credit Approval classification problem.

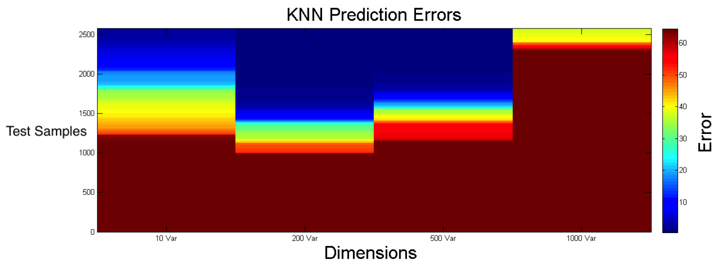


Figure 9. Heat map shows the prediction errors produced by KNN under different number of dimensions. Function F2 (see table 8) was used with different number of dimensions for this analysis to illustrate KNN performance when the dimensionality of the problem increases. The X-Axis shows the KNN prediction for 10, 200, 500, and 1000 variables. This test was done using 2575 test cases with 100 different trees randomly generated. The colour indicate the value of the absolute error of KNN prediction.

further check this, in figure 9, we used function $F2$ (see table 8) with different number of variables (or dimensions). Namely, 10, 200, 500, and 1000 variables. We noticed when the number of variables increases the KNN makes poor predictions. This is was expected because as the dimensionality increases the Euclidean distance between any two points in the Cartesian space becomes obsolete.

4.5.2. Computational Saving In this section, we will try to quantify the computational saving in SSGP against canonical GP. In GP, let the computational cost of a single run calculated in terms of time units denoted as tu.

$$GP^{tu} = \sum (treeSize_i^g \times |T|)$$

where $treeSize_i$ is the size of the i^{th} tree in the g^{th} generation, and $|T|$ is the size of the training set. Hence, the computational cost of canonical GP can be quantified by three parameters: i) the total number of trees (i.e., population size $\times$ number of generations), ii) the size of each tree, and iii) the size of the training set. Now, when we compare this against SSGP:

$$SSGP^{tu} = \sum ([tree_i^g \times treeSize_i \times |T_{sub}|] + KKN(|UnEvaluated|))$$

where $|T_{sub}|$ is sub-training set and $KKN(|UnEvaluated|)$ is the KNN cost when evaluating the difference set between $|T_{sub}|$ and $|T|$ where $|T_{sub}| < |T|$. Note that the cost of KNN is fixed and independent of the tree size. Therefore, as illustrated in the empirical results (see Sections 4.4, 4.2, and 4.3), $SSGP^{tu} < GP^{tu}$ specially when the variables $treeSize_i$ become larger as the search progress.

5. Concluding Remarks and Future Work

This paper proposed a new approach to apply surrogate modelling in GP with expression-tree representation. The new approach saves fitness evaluations through the use of two components. The first component is surrogate model that predicts trees output for a particular input vector x_i based on the similarity between x_i and other input vectors in the training set for which the candidate solution has been already evaluated with. The second component, is a simple linear equation to control the size of a sub-training set that is used to train GP trees. This linear equation allows the size of the sub-training set to dynamically increase or decrease based on the status of the search. The new approach is referred to as SSGP. SSGP allows the GP system to evaluate trees (using a sub-set of training set) and use these as a training set to build a local surrogate model based on KNN. The KNN is used to predict each tree's outputs vector (i.e., semantics) for the remaining unevaluated fitness cases. We have uploaded SSGP code in a public domain so other interested researchers may use the system and compare it with other systems [2].

Among all results reported in the previous sections, the take home message we wish to convey is that SSGP can systematically deliver generalisation performance that is competitive to that of a wide range of GP systems, but SSGP achieves it much faster than its competitors. As an extra advantage, the proposed method for surrogate modelling is very simple to implement and experiment with.

On the other hand, the system's disadvantages are as follows. Currently, SSGP may only be applied on problems with specific search space properties. In particular, each individual should produce one and only one vector value for each input (that is, a given vector of inputs will produce another vector as output). This restriction excludes problems where the GP trees are used directly as programs, like Artificial Ant, or where the produced output is not numeric or vector-like. Another disadvantage of SSGP is that the KNN is based on Euclidean distances which may suffer to produce good approximations in high-dimensional spaces.

For future work, we wish to investigate the following:

- Methods for SSGP to maintain semantic diversity.
- Use of SSGP in Novelty search [14].
- Compare different surrogate models other than KNN in order to predict a program's output vector.

Notes

1. The KNN surrogate has been used to guide the evolution when evaluating the training set only. The evaluation of the validation set was done in the standard way.
2. www.ahmedkattan.com/SSGPcode.zip

References

1. J. R. Anderson, R. S. Michalski, R. S. Michalski, T. M. Mitchell, et al. *Machine learning: An artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.
2. D. L. Donoho. High-dimensional data analysis: the curses and blessings of dimensionality. *American Mathematical Society Conference on Math Challenges of the 21st Century*, August 2000.
3. M. Emmerich, A. Giotis, M. Özdemir, T. Bäck, and K. Giannakoglou. Metamodel-Assisted Evolution Strategies. *Parallel Problem Solving from NaturePPSN VII*, 2439:361–370, 2002.

4. A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.
5. C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 312–321, London, UK, UK, 1994. Springer-Verlag.
6. K. Giannakoglou. Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. *Progress in Aerospace Sciences*, 38(1):43–76, 2002.
7. I. Gonçalves, S. Silva, J. B. Melo, and J. a. M. B. Carreiras. Random sampling technique for overfitting control in genetic programming. In *Proceedings of the 15th European Conference on Genetic Programming*, EuroGP'12, pages 218–229, Berlin, Heidelberg, 2012. Springer-Verlag.
8. Google. Google insights, June 2012. `http://www.google.com/insights/`.
9. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.*, 9(1):3–12, 2005.
10. A. Kattan and E. G. Lopez. Evolving radial basis function networks via gp for estimating fitness values using surrogate models. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2012.
11. A. Keane and N. Petruzzelli. Aircraft wing design using GA-based multi-level strategies. *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, USA 06 - 08 Sep 2000.*, 2000.
12. Y.-H. Kim, A. Moraglio, A. Kattan, and Y. Yoon. Geometric generalisation of surrogate model-based optimisation to combinatorial and program spaces. *Mathematical Problems in Engineering*, 2014, 2014.
13. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.
14. J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, Summer 2011.
15. Y. Lian, M. sing Liou, and A. Oyama. An enhanced evolutionary algorithm with a surrogate model. 2008.
16. D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff. Generalizing surrogate-assisted evolutionary computation. *Evolutionary Computation, IEEE Transactions on*, 14(3):329 –355, 2010.
17. D. Lim, Y. S. Ong, Y. Jin, and B. Sendhoff. Evolutionary Optimization with Dynamic Fidelity Computational Models. *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, pages 235–242, 2008.
18. A. Moraglio and A. Kattan. Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In *EvoCop*, Lecture Notes in Computer Science. Springer, 2011.
19. A. Moraglio and A. Kattan. Geometric surrogate model based optimisation for genetic programming: Initial experiments. Technical report, University of Birmingham, 2011.
20. Y. S. Ong, P. Nair, A. Keane, and K. Wong. Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. *Knowledge Incorporation in Evolutionary Computation, Studies in Fuzziness and Soft Computing Series. Springer Verlag*, 2004.
21. Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modelling. *American Institute of Aeronautics and Astronautics Journal*, 41(4):687–696, 2003.
22. Y.-S. Ong, Z. Zhou, and D. Lim. Curse and blessing of uncertainty in evolutionary algorithm using approximation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2928–2935, 2006.
23. M. L. Repository. Machine learning repository, 2014.
24. M. D. Schmidt and H. Lipson. Coevolution of fitness predictors. *Evolutionary Computation, IEEE Transactions on*, 12(6):736–749, 2008.
25. C. Sun, Y. Jin, J. Zeng, and Y. Yu. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft Computing*, pages 1–15, 2014.
26. Y. Tenne. A Model-Assisted Memetic Algorithm for Expensive Optimization Problems. *Nature-Inspired Algorithms for Optimisation*, pages 133–169, 2009.
27. H. Ulmer, F. Streichert, and A. Zell. Evolution strategies assisted by Gaussian processes with improved preselection criterion. *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*, 1:692–699, 2004.
28. F. Vavak and T. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 192–195, May 1996.

29. Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man and Cybernetics (SMC), part C*, 37(1):66–76, 2005.
30. Z. Z. Zhou, Y. S. Ong, M. H. Lim, and B. S. Lee. Memetic algorithm using multi-surrogates for computationally expensive optimization problems. *Soft Computing Journal*, 11(11):957–971, 2007.