1-1-2015

A Prediction Modeling Framework For Noisy Welding Quality Data

Junheung Park
Wayne State University,

A PREDICTION MODELING FRAMEWORK FOR NOISY WELDING QUALITY DATA

by

JUNHEUNG PARK

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2015

MAJOR: INDUSTRIAL ENGINEERING

Approved By:

_____

Advisor Date

_____

_____

_____

_____

ACHNOWLEDGEMENTS

I am grateful for the tremendous support I have been given by many amazing people throughout my life and education. I highly appreciate my academic advisor, Dr. Kyoung-Yun Kim, for his advice and support in completing this research work. I would also like to acknowledge my appreciation for Drs. Evrim Dalkiran, Richard Darin Ellis, Shiyoung Lu, and Qingyu Yang in providing invaluable advice and recommendations throughout the process of developing and refining my dissertation. My sincere gratitude goes to the entire faculty and staff of the Department of Industrial and Systems Engineering at Wayne State University for their support.

Finally, I would like to thank my family for their endless love and support as I worked to complete this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1. Motivation

The increasing availability of historical data provides various opportunities across different industries. The spectrum of potential opportunities varies from conventional decision support to highly complicated expert knowledge extraction. However, the rapidly growing complexity of data hinders actual implementation of these opportunities. In general, the causes of data complexity can be attributed to various different factors, such as data incompleteness, inconsistency, heterogeneity, high-dimensionality, or rapid change in volume and structure. Fortunately, there has been active research on these data challenges. To achieve the desired outcomes from data, these data challenges should be properly addressed using the right approach or method.

In particular with engineering design and manufacturing applications, during the past few decades, significant efforts have been made to incorporate data into design decision support systems using machine learning and data mining. There has been extensive research conducted on both theoretical and practical aspects in machine learning algorithms and the approaches attempting to resolve data challenges. Every other application favors different types of machine learning algorithms as the data the application faces has different characteristics and requirements. There is no single machine learning algorithm that outperforms others on every possible data. Therefore, the selection of proper machine learning algorithms for the given data is an important problem. As for data challenges, the importance of selecting the right one is as important as it is for the selection of machine learning algorithms.

1.2. Data challenges

Incomplete data, often referred to as missing values, can be classified into three different mechanisms based on the cause of the missingness (Rubin, 1976; Little & Rubin, 1987). The three mechanisms are missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR). Details on these mechanisms are available in (Rubin, 1976; Schafer & Graham, 2002; Graham, 2009; Enders, 2010). Missing values can be filled out by various methods, for example maximum likelihood (ML) and multiple imputation (MI) with no bias as long as these missing values are either MCAR or MAR. However, some methods produce biased estimates in case of MNAR while making it difficult to choose the right imputation method.

As for inconsistency, usually treated as noise, a typical approach is to remove or alter them before constructing prediction model or conduct analysis. Anomaly or outlier detection (Liu & Motada, 2002; Liu, 2010) and instance selection methods (Gamberger *et al.*, 1996; Olvera-López *et al.*, 2010) are usually used for such purposes. However, the cause of sources may vary in different applications so that it may not be appropriate to simply eliminate or alter them in some cases. For instance, in a manufacturing application, the process to acquire data can be complex to configure the desired process environment. More specifically, resistance spot welding (RSW) is widely used in many industries due to its advantages, such as high speed and high volume operations. It is known that a significant inconsistency exists in RSW. Several research works consider predicting welding quality in order to support decisions related to quality monitoring and

material selection (Kim *et al.*, 2003; Pal *et al.*, 2008). Removing or altering these data in such an application may not be desirable since they can be used to extract additional information about explaining the complex nature of the problem.

1.3. Machine learning in design and manufacturing

Machine learning has been applied to utilize available data and to support the complex nature of decision making processes in engineering design and manufacturing domains. These design tasks include conceptual design, design analysis, and design optimization. Conceptual design is one of the early stages in product development, and it is critical since this stage has a significant impacts on the downstream processes of product development (e.g., manufacturing and assembly). There have been a number of research works to support designers in completing certain tasks (Kusiak & Salustri, 2007). For instance, in Venugopal & Narendran (1992), multilayer perceptron (MLP) is used to retrieve design solutions that consist of components used in previous designs including geometric shapes and technological factors. Given these components as inputs, the authors show that designers can retrieve a similar new design solution to improve a final design concept.

Noticeable research works are also conducted in design analysis to discover, understand, and standardize design solutions and processes. Park & Seo (2006) apply MLP to support life cycle assessment of product design alternatives. The dataset include product attributes, lifetime, mode of operations, and energy sources. The constructed MLP model predicts the life cycle assessment of new designs. Sousa & Wallace (2006) employ decision tree (DT) to approximate the life cycle assessment problem. They

consider a set of product attributes, such as energy source, chemicals, and recyclability as input. Classification models are built to classify products into one of the predefined groups and to support in analyzing the relationships between product attributes and their environmental performance. Shieh & Yang (2008) and Yang (2011) present methods to predict customer preference using product form features (e.g., volume, width, shape, and style). SVM and SVR are used respectively for classification and regression to address which customer preference is most satisfied and to predict costumer's preference response value.

Some researchers attempt to use manufacturing information for design decision making. Tang & Chen (2009) aim to achieve an optimal set of parameters for robust processes in sheet metal forming. SVM is used to classify the design space into either feasible or infeasible region being able to provide more accurate predictions compared to the traditional methods. Pan *et al.* (2010) apply SVR to a lightweight B-pillar design problem. Tailor-welded blank structure is used to minimize the weight subject to the constraints of vehicle roof crush and side impacts. SVR approximates the vehicle's roof crush force. The optimal design solutions achieved by the proposed system are promising when compared to the finite element analysis results.

As mentioned earlier, our approach is not to remove or alter noisy data obtained from the manufacturing processes. The presented Meta2 prediction framework aims to construct bagging SVR models, which improves the prediction accuracy on such noisy data with reduced computational cost. By doing so, we expect that the prediction results are more precise and can provide more reliable information about the process.

1.4. Overview

As far as predictive modeling is concerned, the presence of noise in data creates issues, such as over-fitting, which decreases the prediction accuracy on unseen data. In this research, a novel prediction modeling framework, called Meta^2, will be proposed. The aim of this framework is to improve the accuracy of prediction models constructed with the presence of noise in data.

Ensembles, such as boosting and bootstrap aggregating (bagging), are known to improve prediction performance of a learning algorithm. Boosting is usually referred to as a bias reduction approach, whereas bagging can be used to reduce the variance of a learning algorithm. Due to this reason, bagging has been used with unstable learning algorithms, such as decision trees and multilayer perceptron neural network.

In regards to noisy data, research on bagging has proven to improve the prediction accuracy with noisy data (Opitz & Maclin, 1999; Dietterich, 2000; Melville *et al.*, 2004; Khoshgoftaar *et al.*, 2011). Also, recent studies have identified that the prediction accuracy improvement and variance reduction properties of bagging still exist when used with a stable learning algorithm, such as support vector machine (SVM) (Chen *et al.*, 2009; Wang *et al.*, 2009; Kim & Kang, 2012). Therefore, we assume that bagging SVM or support vector regression (SVR), an extension of SVM for regression problems, can improve the accuracy of prediction models on noisy data. To the best of our knowledge, no research has been conducted to identify the property of bagging on both noisy data and regression problems. We assume that bagging SVR will provide an improvement in the prediction accuracy when data used to construct the models consist of noise.

Therefore, the contribution of this research is that it will confirm the applicability of bagging with SVR for regression problems and the prediction accuracy improvement on noisy data.

The prediction accuracy of a SVR model is highly dependent on the selection of its hyper-parameters. Such hyper-parameters usually include the penalty coefficient C, choice of kernel functions, and parameters for the kernel function. In the literature, evolutionary computation (EC) algorithms, such as genetic algorithm (GA) (Wu *et al.*, 2009), particle swarm optimization (PSO) (Lins *et al.*, 2012), and ant colony optimization (ACO) (Zhou *et al.*, 2012), are successfully applied to select the optimal hyper-parameters for SVM and SVR. Generally, they require a large number of candidate solution evaluations to obtain good solutions. The proposed framework employs PSO. Therefore, a candidate solution represents a set of hyper-parameters, whose evaluation is associated with constructing a bagging SVR model using the hyper-parameters. This makes the applicability of an EC algorithm intractable even more so as it increases the computational cost in evaluating (i.e., constructing bagging models for) a large number of candidate solutions. Regarding the previous research on bagging SVM, there has been no related research found in applying EC algorithms to select the hyper-parameters for SVM or SVR included in bagging. To that extent, this research contributes to identifying the applicability of EC algorithms in selecting optimal hyper-parameters.

Meta-modeling, also referred to as surrogates, has been successfully applied to reduce the computational cost when an EC algorithm is associated with a computationally expensive task. For instance, in engineering design, the objective or constraint functions

are often associated with finite element analysis and/or computational fluid dynamic, which both are computationally intensive tasks. The proposed framework uses generalized regression neural network (GRNN) to construct meta-models. Meta-models will be constructed at each iteration of PSO to approximate the fitness function (i.e. objective function). Therefore, the number of candidate solution evaluations will lessen, which will also reduce the computational costs. Some particles in a swarm are re-evaluated with the real fitness function (i.e., constructing bagging SVR) in order to prevent the swarm from moving to the wrong direction due to the approximation errors of these meta-models.

The reasons why GRNN is selected are manifold. The model training process is instance-based. Therefore, we expect to reduce the overall computational requirement further, as long as the training data samples for GRNN are well maintained throughout the PSO iterations. The capability of GRNN has shown to be successful in high dimensional nonlinear problems (Gheyas & Smith, 2010). The only one parameter, called the smoothing factor, is not as sensitive and one can spend less effort in optimizing this smoothing factor value compared to other learning algorithms. Similarly, PSO is chosen due to its efficiency, capability of obtaining quality solutions (Chatterjee *et al.*, 2005; Guo *et al.*, 2008), and of avoiding over-fitting in a similar problem, called full model selection (Escalante *et al.*, 2009). In addition, PSO has been successfully applied to hyper-parameters selection for a single SVM (Lin *et al.*, 2008; Kapp *et al.*, 2009) and SVR (Lins *et al.*, 2012).

1.5 Goals and objectives

Meta^2 prediction modeling framework will be developed for noisy data where the noise cannot be removed or altered before constructing prediction models. The framework will include bagging prediction models using SVR as the base learning algorithm. The hyper-parameters for the SVRs in bagging models are determined by PSO assisted by meta-modeling. GRNN is used to construct the meta-models in the meta-modeling approach. Using this framework, we attempt to construct a bagging SVR model that provides improved prediction accuracy on noisy data. Due to the approximation errors of meta-models, the final prediction model obtained by this framework may provide a lower prediction accuracy than using a regular PSO without meta-modeling. However, we expect to reduce the computational cost in finding such hyper-parameters that is comparable to that of a regular PSO. The main objectives of this research are as follows:

1) Identify the prediction accuracy improvement property of bagging on noisy data using SVR as the base learning algorithm;

2) Develop a computationally efficient meta-modeling approach to assist PSO by approximating the fitness function;

3) Confirm the prediction accuracy of bagging SVR models obtained by Meta^2 with respect to the computational efficiency; and

4) Illustrate how the prediction models constructed by Meta^2 can be used in design activities.

1.6 Organization

For the rest of this report, Chapter 2 reviews bootstrap aggregating and support vector regression. Meta-modeling and our proposed meta-modeling approach will be discussed in Chapter 3 with experimental results. Meta^2 prediction modeling framework will be discussed in Chapter 4 with experimental results. Chapter 5 will conclude this report with future research directions.

CHAPTER 2 CONSTRUCTING BOOTSTRAP AGGREGATING MODELS WITH SUPPORT VECTOR REGRESSION

2.1. Support vector machine and support vector regression

2.1.1. Support vector machine

SVM is a machine learning algorithm for classification problems. Given a training dataset, it seeks to find an optimal hyperplane that classifies data into either the positive or negative class. SVM is also referred to as maximum margin classifier because it aims to maximally separate the positive data samples from the negative. A dataset with n number of samples is represented as $\mathrm{T} \in \{(\mathbf{x}_1, y_1), , \dots, (\mathbf{x}_n, y_n)\}$, where each $\mathbf{x}_i$ is a real numbered vector, $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1, 2, \dots, n$. Each $\mathbf{x}_i$ has its corresponding class label $y_i \in \{-1, +1\}$.

Figure 2.1. illustrates a linearly separable classification problem. Here, $\mathbf{w}$ and b are coefficients that determine the hyperplane and $\boldsymbol{w} \cdot \boldsymbol{x}$ is the dot product. For any data sample $\mathbf{x}_i$, the distance to the optimal hyperplane is $\frac{y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}$. In standard SVM, the

objective is to find an optimal that maximizes the distance between the hyperplane and its closest data instance, which can be formulated as follows:

$$\underset{\mathbf{w},b}{\text{maximize}}(\underset{i}{\text{minimize}}\frac{y_i(\mathbf{w}\cdot\mathbf{x}_i+b)}{\|\mathbf{w}\|}) \tag{2.1}$$

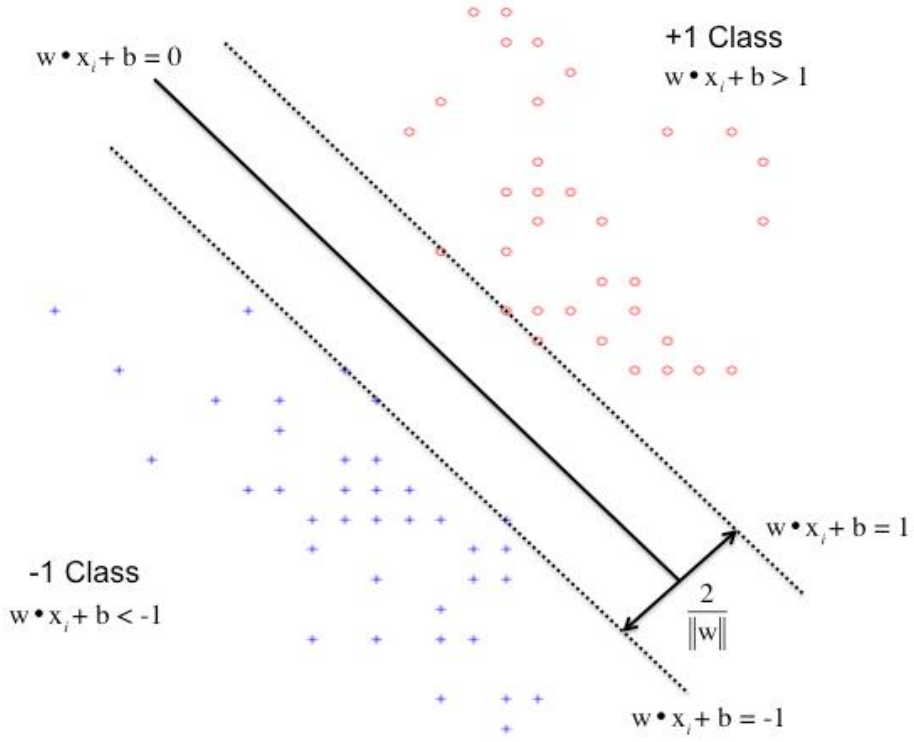


Figure 2.1. Example of a binary classification dataset and optimal hyperplane with the maximum margin

For linearly separable datasets, SVM is an optimization problem of finding a hyperplane $\boldsymbol{w}\cdot\mathbf{x}+b=0$ with the maximum margin that equals 1 (i.e. support vectors). Therefore, the margin is $\rho=\frac{2}{\|\boldsymbol{w}\|}$ since we have two classes. For example, the distance between data samples $\boldsymbol{x}_0$ on $\boldsymbol{w}\cdot\boldsymbol{x}_i+b=-1$ and $\boldsymbol{x}_1$ on $\boldsymbol{w}\cdot\boldsymbol{x}_i+b=1$ is $\frac{1}{\|\boldsymbol{w}\|}$. These data

samples closest to the hyperplane are called support vectors. In addition, for any data instance $\boldsymbol{x}_i$, we know that $\boldsymbol{w} \cdot \boldsymbol{x}_i + \mathrm{b} > 1$ if $y_i = 1$, otherwise $\boldsymbol{w} \cdot \boldsymbol{x}_i + b < -1$. This can also be written as $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1$. Now we can formulate this into an optimization problem as follows:

$$\begin{aligned} &\underset{\mathbf{w},b}{\text{maximize}} && \frac{2}{\|\mathbf{w}\|} \\ &\text{subject to} && y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n \end{aligned} \tag{2.2}$$

Maximizing $\frac{2}{\|\boldsymbol{w}\|}$ is equivalent to minimizing $\frac{1}{2}\|\boldsymbol{w}\|^2$. Thus, Equation (2.2) can be converted to a minimization problem as follows:

$$\begin{aligned} &\underset{\mathbf{w},b}{\text{minimize}} && \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to} && y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1\,, \quad i = 1, \ldots, n \end{aligned} \tag{2.3}$$

Equation (2.3) is a quadratic programming problem. In practice, Equation (2.3) is usually converted to its dual formulation using Lagrange multipliers. By doing so, one can reformulate the problem for linearly non-separable and nonlinear support vector machines (Ivanciuc, 2007). The Lagrange dual formulation of Equation (2.3) can be defined as follows:

$$\begin{aligned} L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_{i=1}^{n} \alpha_i \\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \mathbf{w} \cdot \mathbf{x}_i - \sum_{i=1}^{n} \alpha_i y_i \mathrm{b} + \sum_{i=1}^{n} \alpha_i \end{aligned} \tag{2.4}$$

Then, using the Karuch-Kuhn-Tucker (KKT) conditions for the above Lagrange function and solving the Wolfe dual problem of Equation (2.4) to these KKT conditions, one can solve the SVM problem (Ivanciuc, 2007). The KKT conditions for Equation (2.4) are as follows:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{2.5}$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.6}$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \alpha_i} \rightarrow g_i(\mathbf{x}) = 0 \rightarrow \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0, i = 1, \dots, n \tag{2.7}$$

$$\alpha_i \geq 0, i = 1, \dots, n \tag{2.8}$$

By plugging in Equation (2.5) and (2.6) to the Lagrange function, Equation (2.4), the Lagrange dual problem can be defined as follows:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ \text{subject to} \quad & \alpha_i \geq 0, \, i = 1, 2, \dots, n \\ & \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.9}$$

Note that each data sample has its corresponding Lagrange multiplier α_i. Once the above SVM problem is solved, data samples with $\alpha_i > 0$ are identified as support vectors. Now, one can compute the vector $\boldsymbol{w}$ and threshold b to obtain the optimal hyperplane using Equation (2.5) and (2.7). Using the hyperplane, a new data $\boldsymbol{x}_0$ can be predicted as class +1 if $\boldsymbol{w} \cdot \boldsymbol{x}_0 + b > 0$, otherwise class -1.

So far, the SVM formulation has been reviewed for linearly separable datasets. In case of linearly non-separable datasets, the slack variable ξ is introduced to penalize data samples that are not correctly classified. After adding the slack variable ξ for each data sample and a penalty coefficient C to the objective function, Equation (2.3) can be modified for linearly non-separable datasets. Such modification with respect to support vector regression (SVR) is introduced in Section 2.1.2. Equation (2.11). Once the SVM optimization problem for linearly non-separable datasets is formulated, the optimization problem can be transformed to its Wolfe dual problem and used to achieve the support vectors. The transformation can be done in the same way described above for linearly separable cases. The SVM formulations for linearly separable and linearly non-separable are called hard margin and soft margin linear SVM respectively. The formulation of soft margin linear SVM is as follows:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ \text{subject to} \quad & 0 \le \alpha_i \le C,\ i = 1, 2, \ldots, n \\ & \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.10}$$

Similarly for nonlinear datasets, the Wolfe dual problem of either hard margin or linear margin linear SVM formulation can be used with a kernel function $K(\,)$. The kernel function can be applied to the dot product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ in the objective function of Equation (2.10). This is called the kernel trick. The formulations and proofs of soft margin linear, hard margin nonlinear, and soft margin nonlinear SVM are available with

in some of tutorial articles (Burges, 1998; Ivanciuc, 2007). Also, detailed description of the theory behind SVM and proofs can be found in Vapnik (1998).

2.1.2. Support vector regression

SVM can be applied to classification problems. However, there are many applications where it is required to predict continuous values (i.e., regression problems) instead of class labels. These applications include regression, time series analysis, etc. Similar to classification datasets, suppose we have a dataset with n number of samples $\mathrm{T} \in \{(\mathbf{x}_1, y_1), , \dots, (\mathbf{x}_n, y_n)\}, \}$, where each $\mathbf{x}_i$ represents each data sample (i.e., $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1, 2, \dots, n$) and each $\mathbf{x}_i$ has its corresponding response value $y_i \in \mathbb{R}$. The aim is to construct a prediction function $\widehat{f}(\boldsymbol{x})$ to approximate the original function $f(\boldsymbol{x})$. Vapnik (1995) develop so-called ε-support vector regression (SVR) by extending the notion of the maximum margin hyperplane in SVM to regression problems.

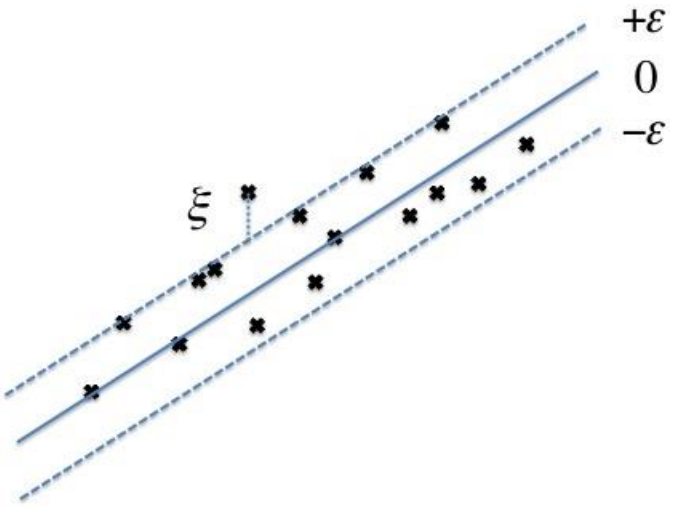


Figure 2.2. ε precision and slack variable ξ in ε-SVR

ε-SVR attempts to minimize the errors between the target and prediction by finding an optimal hyperplane such that the prediction error for each training data does not exceed the ε precision. This allows errors that are less than ε. This assumption may not be true in many datasets, meaning that the solution is not feasible. In many real-world datasets, this is not possible due to the variability and noise. In order to address this issue, the notion of soft margin can be used the same way as in SVM by introducing the slack variable. Figure 2.2. illustrates these notions of ε precision and slack variable ξ.

Accordingly, the hard margin and soft margin linear SVR can be formulated in the same way as in SVM (Smola & Schölkopf, 2004).

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i - (\mathbf{w}\cdot\mathbf{x}_i + b) \le \varepsilon\,, \quad i = 1,\dots,n \\ & (\mathbf{w}\cdot\mathbf{x}_i + b) - y_i \le \varepsilon\,, \quad i = 1,\dots,n \end{aligned} \tag{2.11}$$

Equation (2.11) represents the hard margin linear SVR. The soft margin linear SVR can be formulated as follows:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - (\mathbf{w}\cdot\mathbf{x}_i + b) \le \varepsilon + \xi_i\,, \quad i = 1,\dots,n \\ & (\mathbf{w}\cdot\mathbf{x}_i + b) - y_i \le \varepsilon + \xi_i^*\,, \quad i = 1,\dots,n \\ & \xi_i, \xi_i^* \ge 0, i = 1,\dots,n \end{aligned} \tag{2.12}$$

These SVR optimization problems are usually transformed to the Wolfe dual problem and the support vectors are obtained in the same manner in SVM. Equation (2.13) represents the Wolfe dual formulation of Equation (2.12).

$$\underset{\alpha,\alpha^*}{\text{maximize}} \quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\mathbf{x}_i\cdot\mathbf{x}_j-\varepsilon\sum_{j=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{j=1}^{n}y_i(\alpha_i-\alpha_i^*)$$

$$\text{subject to} \quad 0\le\alpha_i,\alpha_i^*\le C,\ i=1,2,\dots,n \qquad (2.13)$$

$$\textstyle\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0$$

Also, using the support vectors and KKT conditions, one can compute **w** and b to predict new data. The kernel trick can be used to the dual problem in order to solve nonlinear regression problems. Detailed proofs and derivation are available in a tutorial on SVR in Smola & Schölkopf (2004).

2.2. Bootstrap aggregating

2.2.1. Constructing bootstrap aggregating models

Bagging (Breiman, 1996) is one of the popular ensemble methods along with boosting. Suppose we are given a dataset T $\in\{(\mathbf{x}_1,y_1),,\dots,(\mathbf{x}_n,y_n)\}$ where y_i is either the class label or continuous valued response corresponding to the *i*th data sample $\boldsymbol{x}_i$. The aim is to construct a prediction function $\hat{f}_{bag}(\mathbf{x})$. Bagging generates multiple bootstrapped datasets from the original dataset T by randomly drawing samples with replacement. Let $l=1,\dots,L$ be the number of bootstrapped datasets. T^l is the *l*th bootstrapped dataset sampled with replacement from the original dataset T. The size of each of these bootstrapped datasets, n', is usually the same as the original training dataset. As a result of bootstrapping, some data samples can appear multiples times in each bootstrapped dataset while some may not be included in it at all. The probability that each

training data sample is selected at least once is equal to $1 - \left(1 - \frac{1}{n'}\right)^{n'}$ (Bauer & Kohavi, 1999). Therefore, when $n' = n$, the probability is about 63.2%

Then, using a base learning algorithm such as decision tree (DT), neural network (NN), and support vector machine (SVM), one can construct each prediction function $\hat{f}^l(\mathbf{x})$ corresponding to the *l*th bootstrapped dataset. Once these prediction functions are constructed the bagging prediction function $\hat{f}_{bag}(\mathbf{x})$ is obtained by combining the results of $\hat{f}^l(\mathbf{x})$ as follows:

$$\hat{f}_{bag}(\mathbf{x}) = \varphi(\hat{f}^l(\mathbf{x})) \tag{2.14}$$

where $\varphi(\,)$ is an aggregating function. There are many number of different aggregating functions proposed in the literature.

We report in Table 2.1. several aggregating functions introduced in Polikar (2012), which are applicable to regression problems. For classification problems, the results of these multiple prediction functions $\hat{f}^l(\mathbf{x})$ can be aggregated, for example, using a majority voting aggregating function, where the final class label is the one that wins the most vote from $\hat{f}^l(\mathbf{x})$. Figure 2.3. shows the pseudo code of bagging.

Input: training dataset T, learning machine M, number of bootstrap datasets *L*, size of a bootstrap dataset n'
1: for l = 1 to L
2: T_l = bootstrapped sample from T
3: $\hat{f}^l = M(T_l)$
4: $\hat{f}_{bag}(\mathbf{x}_0) = \varphi\left(\hat{f}^l(\mathbf{x}_0)\right)$
Output: Bagging predictor $\hat{f}_{bag}$

Figure 2.3. Bagging Pseudo code

Table 2.1. Aggregating functions for regression (Polikar, 2012)

Mean	$\frac{1}{L}\sum_{l=1}^{L}\hat{f}^{l}(\mathbf{x})$
Sum	$\sum_{l=1}^{L}\hat{f}^{l}(\mathbf{x})$
Weighted sum	$\sum_{l=1}^{L}w_l\hat{f}^{l}(\mathbf{x})$, where w_l is the corresponding weight to $\hat{f}^{l}$.
Product	$\prod_{l=1}^{L}\hat{f}^{l}(\mathbf{x})$
Maximum	$\max_{l=1,\ldots,L}\{\hat{f}^{l}(\mathbf{x})\}$
Minimum	$\min_{l=1,\ldots,L}\{\hat{f}^{l}(\mathbf{x})\}$
Median	$\operatorname{med}_{l=1,\ldots,L}\{\hat{f}^{l}(\mathbf{x})\}$
Generalized mean	$\left(\frac{1}{L}\sum_{l=1}^{L}\hat{f}^{l}(\mathbf{x})^{\alpha}\right)^{1/\alpha}$ $\alpha \to -\infty \Longrightarrow$ Minimum $\alpha \to \infty \Longrightarrow$ Maximum $\alpha \to 0 \Longrightarrow$ Geometric mean $\alpha \to 1 \Longrightarrow$ Mean

2.2.2. Selection of base learning algorithm in bootstrap aggregating

In Breiman (1996), it is noted that the base learning algorithm has to be unstable, which means small changes in the training sets may result in a large difference in the final prediction function. Such unstable ones are DT and NN. In regards to noisy data, Dietterich (2000) conduct a series of experiments on 33 classification datasets to compare the prediction accuracy of bagging, boosting, and randomization. DT is used as the base learner. Boosting and randomization provide better results than bagging when there is little noise. Bagging is the best among these methods when classification noise is added while producing more diverse classifiers. Similarly, DT and Naïve-Bayes algorithms are

examined in Bauer & Kohavi (1999). They report that boosting outperforms bagging and its variant ensemble methods using both DT and Naïve-Bayes. Bagging does not seem to have a significant effect in improving the prediction accuracy with Naïve-Bayes. However, the conducted experiments do not address the prediction accuracy when noise is present in the dataset.

In addition, bagging and boosting are evaluated for DT and NN in Opitz & Maclin (1999). It is noted that bagging DT or NN always outperforms a single base learner either DT or NN. Their experimental results also show bagging provides a more consistent prediction performance than boosting when noise is present in data. Melville *et al.* (2004) emphasize on the performance of ensemble methods when different types of imperfection including missing data and classification noise are present in data. In case of noise, different levels of noise are added to the datasets considered and bagging provides the best prediction accuracy. Note that DT is used as the base learning algorithm of the ensemble methods compared. A recent study (Khoshgoftaar *et al.*, 2011) reveals that bagging is better than boosting when data contain both class imbalance and noise. The study also suggests to use bagging by sampling without replacement instead of sampling with replacement in that case. DT, Naïve-Bayes, and rule-based learner are considered the base learning algorithm in the study.

The previous experiments mentioned above suggest that the base learning algorithm of bagging should be unstable. However, research has been conducted on identifying the effectiveness of bagging with SVM, which is not known as an unstable algorithm. The following Table 2.2. briefly summarizes these research in terms of their

objectives, results, and the hyper-parameters selection approach. All of the research focuses on classification problems, while most of them compare boosting and/or bagging SVM to a single SVM. Bagging SVM always improves a single SVM in all cases, which proves that SVM is not an inappropriate choice for bagging. On the other hand, boosting SVM does not always outperform a single SVM. A recent study in Kim & Kang (2012) attempts to select optimal classifiers in boosting and bagging called CO-boosting and CO-bagging respectively. DT, NN, and SVM are considered as the base learner. Regular bagging SVM does not outperform bagging DT or NN. However, CO-bagging, where optimal SVMs in a bagging are selected, provides the best results on the dataset considered.

Regarding noise, Valentini (2005) reports that bagging and random aggregating (RA) perform similarly outperforming a single SVM. However, without noise in the dataset, RA SVM generally outperforms bagging SVM. We claim that previous research has paid less attention to regression problems than classification and noisy data as well. Again, these two areas are one of the objectives in this research. We assume the applicability of bagging SVM can be extended to bagging SVR since both SVM and SVR share the core theoretical properties in determining the maximum margin hyperplane explained in Section 2.1.

As shown in Table 2.2., we focus on selecting the hyper-parameters in bagging SVR. As far as the hyper-parameters selection is concerned in a bagging model, one needs to decide whether all the SVRs in a bagging should use the same set of hyper-parameters or not. Even though Chen *et al.* (2009) use different sets for each SVM and

show its effectiveness in the prediction performance, the majority of related research apply the same set of hyper-parameters to all the SVMs. We consider the latter approach in this research.

Table 2.2. SVM ensembles related research on their objectives, results, and the hyper-parameters selection approach

Same hyper-parameters applied to all the learners in an ensemble
Kim *et al.* (2002)
• Single and bagging SVM with different aggregating functions on three datasets • Bagging SVM outperforms a single SVM • A predefine set of hyper-parameters
Kim *et al.* (2003)
• Single, bagging, and boosting SVM with different aggregating functions on three datasets • Bagging SVM outperforms a single SVM while boosting SVM slightly outperforms bagging SVM • A predefine set of hyper-parameters
Valentini & Dietterich (2003)
• Single, bagging, and new bagging SVM on seven datasets • Bagging is better than a single SVM while their proposed bagging outperforms both • Different sets of hyper-parameters are tested using a grid search
Valentini (2005)
• Single, bagging, and random aggregating (RA) SVM on seven datasets • Both bagging and RA SVM outperform a single SVM while RA SVM provides a larger improvement than bagging SVM. In case of noisy data, RA and bagging SVM provide a similar level of improvement • Different sets of hyper-parameters are tested using a grid search
Pal (2008)
• Single, boosting, and bagging SVM on land cover classification dataset • Bagging outperforms others • A set of hyper-parameters known to work well for the dataset in a previous research
Wang *et al.* (2009)

• Single, bagging, and boosting SVM on 20 datasets and an industrial case of gear defect detection dataset • Bagging seems to be the most appropriate ensemble for the most datasets considered providing relatively better prediction accuracy • Several sets of predefined hyper-parameters are tested
Kim & Kang (2012)
• Single, bagging, boosting, and cover optimized bagging and boosting using DT, NN, and SVM on bankruptcy prediction dataset. • Bagging SVM outperforms single SVM but not boosting SVM while bagging other base learners outperform boosting. • Different sets of hyper-parameters are tested using a grid search

Different hyper-parameters applied to each learner in an ensemble
Chen *et al.* (2009)
• Single, bagging, and boosting SVM on traffic incident detection dataset using different performance measures • Bagging outperforms others in several performance measures considered • Randomly generate hyper-parameters for each classifier in an ensemble

2.3. Hyper-parameters selection approaches for single support vector machine and support vector regression

The selection of hyper-parameters in SVM and SVR is crucial and directly related to the prediction accuracy of constructed models. In ε-SVR, the value of ε determines the level of accuracy as described earlier. If ε is too large, the constructed models may under-fit failing to include the target values in the ε precision. On the other hand, a too small ε can over-fit the data. Typically, they hyper-parameters one should determine also include the penalty coefficient ε, kernel function, and corresponding kernel parameters. This is called hyper-parameters or model selection problem in the literature. Table 2.3. below shows four commonly used kernels.

Table 2.3. Four typical kernel functions for SVM and SVR

Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\gamma(\mathbf{x}_i^T \mathbf{x}_j + 1)\right)^d$
Radial basis function	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp -\gamma \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^d$
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + d)$

In this section, we review the related literature and define a classification of approaches to the hyper-parameters selection problem. This is to identify advantages and disadvantages of the existing approaches. We define two different types of approaches based on our literature review; i) Computational approach and ii) Analytical approach.

Grid search is one of the conventional approaches to the hyper-parameters selection problem. In grid search, each hyper-parameter is assigned with a search space (e.g., using a maximum, minimum, and interval). Then each candidate parameter is directly used to construct a prediction model using SVM or SVR. Finally, one can select the candidate hyper-parameters that provides the best prediction accuracy. The wider and finer search space, the better quality hyper-parameters one can obtain. Obviously, this comes at a higher computational cost. However, as far as the computational cost is not concerned, grid search is still being applied in many applications. For instance, Kavaklioglu (2011) employs SVR in order to model the electricity consumption of Turkey. The choice of kernel functions is limited to radial basis function kernel, which leaves less computational burden. Although the results of SVR are not compared in terms of the performance of grid search or prediction accuracy, it is concluded that SVR using grid search is sufficient for the electricity consumption of Turkey.

We list a number of related research and their summary in each category as follows:

i) Computational approach: There has been research that employs evolutionary computation algorithms, including genetic algorithm (GA) and particle swarm optimization (PSO). Chen (2007) uses a real-valued GA in order to construct an optimal SVR for turbochargers reliability dataset. The constructed SVR model shows better prediction accuracy compared to several other machine learning algorithms such as multilayer-perceptron (MLP). In Lin *et al.* (2008), PSO is considered to find the optimal subset of features and hyper-parameters for classifiers constructed using 17 classification datasets. The experimental results obtained by PSO are compared to that of grid search and GA and show that PSO provides better prediction accuracy. Kapp *et al.* (2009) modify PSO in order to further reduce its computational cost. The authors compare their proposed method with the results achieved by grid search and regular PSO on five classification datasets. The proposed method provides a comparable quality solution to the regular PSO with less computational requirements.

Aforementioned research only considers the radial basis function (RBF) kernel. Wu *et al.* (2009) hybridize a real-valued and integer-valued GA so that it will be able to find the optimal kernel function and the hyper-parameters accordingly for SVR in an electrical daily load prediction application. The experimental results demonstrate improved prediction accuracy using their proposed GA compared to a regular GA with the selection of kernel functions set to the RBF only. Similarly, GA and PSO are

successfully applied to find the optimal hyper-parameters in a product form design and reliability application respectively in Yang & Shieh (2010) and Lins *et al.* (2012).

Small world optimization (SWO) is coupled with tabu search called TSWO in Mao *et al.* (2012). It is claimed that PSO and GA can find the global optimum and yet they have a possibility of premature convergence when the optimization problem is complex. Their proposed method TSWO shows that it avoids premature convergence and provides better performance than PSO and GA when tested on several artificial datasets generated from multimodal functions including sine. Zhou *et al.* (2012) employ ant colony optimization (ACO) for NOx emission modeling using SVR. The hyper-parameters found by ACO are used to construct SVR models and compared to that of grid search and MLP as well.

In addition, other than evolutionary computation algorithms, a few computational approaches are proposed in Jeng (2005) and Huang *et al.* (2007) and demonstrate their capability of finding quality solutions in an efficient manner.

ii) Analytical approach: One of the representative analytical approaches is gradient-based. Such gradient-based approaches are developed in Bengio (2000), Chapelle *et al.* (2002), Ayat *et al.* (2005), Chang & Lin (2005), Moser & Serpico (2009). These proposed approaches have shown their capabilities in improving accuracy and computational efficiency. However, they require the objective function to be differentiable with respect to the hyper-parameters (Kapp *et al.*, 2009), high chance of falling in a local minima (Huang *et al.*, 2007; Kapp *et al.*, 2009). Therefore, due to

these characteristics, it may not be trivial in many practical applications (Lins *et al.*, 2012).

2.4. Summary

A review on bagging and support vector regression is given in this section. Notations and equations are discussed. Also, related research on how to select the hyper-parameters for SVR are introduced. Computational and analytical approaches are defined each of which includes EC algorithms and gradient-based optimization algorithms. Regarding bagging, selecting the base learning algorithm, bootstrapped dataset, and aggregating functions are reviewed. More importantly, we identify whether or not to use the same set of hyper-parameters for SVRs included in a bagging model.

CHAPTER 3 META-MODELING FOR FITNESES FUNCTION APPROXIMATION TO ASSIST EVOLUTIONARY COMPUTATION

3.1. Introduction

Evolutionary computation (EC) aims to find optimal solutions for various types of optimization problems. EC includes genetic algorithm (GA), genetic programming (GP), estimation of distribution algorithm (EDA), and swarm intelligence (SI). Examples of SI include particle swarm optimization (PSO) and ant colony optimization (ACO). The term EC is often treated as the same as evolutionary algorithm (EA) in the literature (Zhang *et al.*, 2011). These EC algorithms have been applied in various optimization problems throughout different domains such as bioinformatics (Pal *et al.*, 2006), machine learning

(Zhang *et al.*, 2011), and engineering problems (Arciszewski & Jong, 2001; Fleming & Purshouse, 2002).

EC algorithms in general require a large number of fitness function evaluations on candidate solutions as the population or generation evolves. In many applications, these fitness functions can be associated with a computationally expensive analysis or simulation. For instance, in a complex engineering design problem, such as an aircraft design optimization, the design simulation processes are computationally expensive where complex analyses such as finite element analysis and computational fluid dynamics are required (Wang & Shan, 2007). One of the approaches to improve the computational efficiency is meta-modeling, also called surrogates, where the fitness function can be approximated to reduce the number of fitness function evaluations.

Meta-modeling can be defined as a model of the model (Kleijen, 1986). During the past decade, there has been a large number of meta-modeling research works proposed in the literature. These research works consider various types of meta-modeling algorithms, model construction schemes, and EC algorithms as well. Several review articles are available on the meta-modeling techniques in Jin *et al.* (2001), Jin (2005), and Jin (2011). The typical choice of meta-modeling algorithms includes polynomial regression (PR), multilayer perceptron (MLP) neural network, kriging, and radial basis function (RBF) network. A recent, related experiment was conducted on an aerodynamic design problem using evolutionary programming and support vector regression (Andrés *et al.*, 2012). In regards to EC algorithms, GA (Dias *et al.*, 2013), PSO (Sun *et al.*, 2014), and differential evolution (DE) (Park & Lee, 2014) are considered in recent studies.

As far as the meta-modeling algorithm is concerned, the choice is not limited to those mentioned above. For instance, the capability of generalized regression neural network (GRNN) for meta-modeling is examined and shows promising results in Fang-shu & Jian-Chao (2009). GRNN has several advantages to improving the efficiency in constructing meta-models. GRNN's model training process is an instance-based approach. Therefore, one can expect to reduce the computational cost caused by the iterations, such as MLP, RBF network, and SVR. This is a considerable advantage over other algorithms such as kriging, which is known to have a high computational cost in the model construction. Updating kriging models with new data samples is not trivial (Jin, 2011). In addition, GRNN requires only one parameter other than the selection of distance measure (e.g., Euclidean). This parameter is called the smoothing factor and known insensitive (Gheyas & Smith, 2010). As for the EC algorithm, PSO is chosen because of its advantages. Most of all, the implementation of PSO is straightforward, the number of parameters is less than many other EC algorithms, and efficiency with reasonable quality of solutions (Zhang *et al.*, 2000; Mendes *et al.*, 2002; Chatterjee *et al.*, 2005; Guo *et al.*, 2008; Escalante *et al.*, 2009).

To the best of our knowledge, less attention has been given to meta-modeling using GRNN although it offers positive advantages applicable to meta-modeling as mentioned above. Therefore, we propose Meta-modeling Using GRNN and PSO (called MUGPSO) and aim to identify its capability as a meta-modeling algorithm in this research. For that purpose, we maintain the simplest possible meta-modeling scheme where one global meta-model constructed by GRNN is maintained and updated

throughout the run. The global meta-model is used to approximate the fitness function of candidate solutions (i.e., particles) in the entire swarm. MUGPSO is tested on various benchmark problems from the literature with different characteristics and compared with the performance of other recent meta-modeling algorithms.

In this research, we employ MUGPSO to help improve the overall computational efficiency in selecting the optimal hyper-parameters for bagging SVR models. However, the applicability of meta-modeling in evolutionary computation is not limited to the fitness function approximation. Population initialization, cross-over, mutation, and local search can be replaced or assisted by meta-modeling (Jin, 2011). In this chapter, we introduce meta-modeling models, kriging, PR, RBF networks, and GRNN. Techniques related to meta-model training such as data sampling and evolution control are also included.

3.2. Related work

3.2.1. Meta-models for fitness function approximation

Several previous research are available in relation to meta-modeling the fitness function approximation in evolutionary computation. Jin *et al.* (2001) study four meta-modeling techniques, KG, PR, RBF, and Multivariate Adaptive Regression Splines (MARS). These meta-modeling techniques are tested on 14 different mathematical and engineering test problems where the degree of nonlinearity, dimension, and noise is different. Their performance are measured and compared in terms of accuracy, efficiency, transparency, and simplicity. RBF shows the best performance among the four meta-modeling techniques in terms of accuracy. RBF has the lowest impact on the sample size

and noise. Another advantage of RBF is the simplicity to implement while KG and MARS are not due to their parameters. MARS does not perform well on difficult problems where the degree of nonlinearity is high, small number of samples are available to train, and the number of input features is high.

Similarly in Lim *et al.* (2007), PR, KG, RBF and multilayer-perceptron (MLP) are considered. Meta-models are built using these four techniques and included in a memetic algorithm. Four test functions are tested in order to compare the performance of the meta-modeling techniques. The results compared to that of a regular genetic algorithm (GA) reveal that PR and KG seem to provide more robust performance for the four test functions.

We introduce kriging (KG), radial basis function (RBF), and polynomial regression (PR) for meta-modeling methods to approximate the fitness function. It should be noted that the selection of the approximation method is not limited to these three introduced here. Based on the complexity of fitness function, one can consider one of the simplest methods, such as *k* nearest neighbors. In Clark *et al.* (2005), support vector regression is compared to KG, RBF, and PR over a number of test problems showing a better performance.

3.2.2. Data sampling and evolution control techniques for meta-modeling

One of the main objectives of meta-modeling in evolutionary computation is to reduce the computational cost (e.g., the actual fitness evaluation of computationally expensive function for candidate solutions in EC). It is difficult to construct an approximation model that guarantees a global optimal due to several reasons such as the

high dimensionality and limited number of training samples (Jin, 2005). Therefore, the data samples from which the meta-model is constructed have to be carefully chosen in order for the approximate model to be accurate as much as possible.

It is obvious that more data samples will be likely to result in a better approximate model. Hence, it provides EC algorithms with a higher chance of obtaining better quality solution (i.e., close to the global optimal). However, this requires more computational effort in evaluating them with the original fitness function, which is assumed to be an expensive one.

The initial data samples can be generated and then need to be evaluated for their fitness function values in order to construct an initial approximate model. In some applications, domain expert or history data are available to provide such initial data samples. These cases fall into off-line sampling and training techniques. As the EC algorithm continues throughout the iterations, candidate solutions at each iteration are generated from the previous iteration using some sort of operators or rules. For instance, a GA generate new set of candidate solutions at each iteration applying operators called mutation and cross-over on the previous candidate solutions at the previous iteration. Then, these candidate solutions have to be either evaluated or approximated.

Evolution control concerns managing which candidate solution to evaluate with the original fitness function and to approximate with the approximate model. Using the approximate model together with the original fitness function can improve the quality of final solution achieved by the EC algorithm. Obviously, the computational cost increases as more candidate solutions are needed to evaluate with the original fitness function. In

this process, new data samples that are evaluated with the original fitness function are obtained. Using these data samples, the initial approximate model can be retrained to improve its accuracy, which falls into on-line updating with evolution control. One can exclusively employ an off-line sampling and training technique or include an on-line updating with evolution control technique.

One of the most popular off-line sampling and training techniques is design of experiments (DOE) based techniques. These DOE techniques include Latin hepercube (LH) and central composite design (CCD). Given the dimension of the problem d (i.e., the number of input features), LH splits each input feature range into m strata of equal probability $1/m$ (McKay *et al.*, 1979). Then m values are randomly distributed with one from each stratum and they are randomly permuted forming the final set of data samples. CCD, along with Box-Behnken design, is a widely used DOE technique to estimate a second-order polynomial approximation (Wang & Wan, 2009). A CCD consists of a 2^d factorial points, star points, and center points. An illustrative example for a two-dimensional problem is shown in Figure 3.1.

Detailed reviews on DOE techniques including LH and CCD are available in Robinson *et al.* (2004) and Hibbert (2012). Besides DOE techniques, off-line data sampling can be achieved by Monte-Carlo and active learning methods.

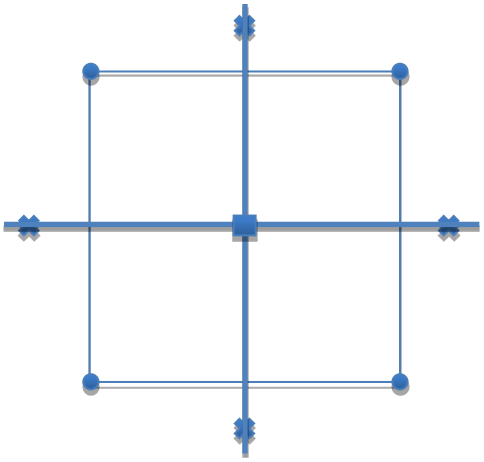


Figure 3.1. An example of a central composite design for a two-dimensional problem. (Circle, cross, square points indicate factorial, star, and center points respectively.)

On the other hand, various approaches are available for on-line updating with evolution control. The most straightforward approach is to evaluate candidate solutions that may have a good fitness function value (Jin, 2011). To this extent, the best candidate solution, or several good solutions, at each iteration can be evaluated with the original fitness function and the approximate model can be retrained with the data sample included. The best candidate solution can be assumed to be the one that has the best fitness function value approximated by the meta-model. These types of approaches are referred to as an individual-based approach. On the other hand, one can also consider generation-based approaches where all the candidate solutions in a fixed number of iteration are evaluated. In this research, we focus on individual approaches.

Several research employ individual-based approaches by clustering the candidate solution in each generation and choosing ones that are close to the centers (Kim & Cho, 2001; Jin & Sendhoff, 2004; Gomide, 2006) or the best ones in each cluster (Graning *et*

al., 2007). Instead of choosing the best candidate solution, randomly choosing a number of candidate solutions is also possible. However, this random approach has not shown to outperform the best approach (Jin, 2005).

Besides choosing best ones and random selection, another criterion is the degree of approximation uncertainty (Jin, 2011). Because the accuracy of the approximate model constructed by meta-models greatly affects the success in finding a good optimal solution, such candidate solutions that have high degree of approximation uncertainty have more potential to improve the approximate model.

3.2.3. Meta-modeling with particle swarm optimization

Fitness function approximation using meta-modeling has been given much attention in the literature during the past decade. However, the use of meta-modeling for PSO is relatively less than other EC algorithms (Sun *et al.*, 2013; Sun *et al.*, 2014). Here, we focus on the related research to meta-modeling for PSO. Recent meta-modeling research for GA and DE include Dias *et al.* (2013) and Park & Lee (2014), respectively.

Reyes-Sierra & Coello (2005) propose four different fitness approximation approaches based on the closest particles and apply in a multi-objective PSO. Hendtlass (2007) defines a reliability measure on each particle and estimate the fitness based on the fitness inheritance. GRNN is applied in Fang-shu & Jian-Chao (2009) to support PSO by approximating the fitness function evaluation. Praveen & Duvigneau (2009) propose using meta-models constructed by RBF network and examine the performance in an aerodynamic shape design application, where the real fitness function is associated with

CFD. Similarly, Bird & Li (2010) and Parno *et al.* (2012) consider meta-modeling using polynomial regression and kriging.

More recently, Sun *et al.* (2013) propose a new fitness approximation approach for PSO based on fitness inheritance. An ensemble of RBF networks and polynomial regression is proposed in Tang *et al.* (2013) and tested on several benchmark functions and engineering design problems. In Ren *et al.* (2013), GRNN is considered for fitness function approximation and tested on a few benchmark functions. Regis (2014) develops a meta-modeling framework, which uses PSO and RBF network. In the framework, each particle in the swarm considers multiple trial positions and the most promising particle is chosen using the RBF meta-model. Sun *et al.* (2014) propose a meta-modeling framework called two-layer surrogate assisted particle swarm optimization (TLSAPSO). TLSAPSO employs two different types of meta-model constructed by RBF networks. They are called the global and local meta-models where the global meta-model is constructed based on the whole swarm and expected to smooth out the local optimum. The local meta-models are constructed for each particle and aim to approximate the local fitness landscape. These two types of models are selectively used throughout the iterations based on the accuracy. The aforementioned algorithms are compared with the presented MUGPSO and discussed later sections in this section.

3.3. Meta-modeling algorithms

3.3.1. Kriging

A kriging model can be written to represent the original function as follows:

$$f(\mathbf{x}) = g(\mathbf{x}) + Z(\mathbf{x}) \tag{3.1}$$

, where $f(\mathbf{x})$ is a kriging model, $g(\mathbf{x})$ is a global model of the original function, and $Z(\mathbf{x})$ represents a local deviation from the global model $g(\mathbf{x})$, which is usually a Gaussian random function with zero mean and non-zero covariance (Jin, 2005). Assuming that the global model $g(\mathbf{x})$ is a polynomial, Equation (3.1) can be rewritten as follows:

$$f(\mathbf{x}) = \beta + Z(\mathbf{x}) \tag{3.2}$$

, where β represents the underlying coefficients of the polynomial. The covariance $Z(\mathbf{x})$ can be represented as follows:

$$Cov\big[Z(\mathbf{x}_i), Z(\mathbf{x}_j)\big] = \sigma^2 R\big(\mathbf{x}_i, \mathbf{x}_j\big) \tag{3.3}$$

, where σ^2 is the process variance, R is the correlation between any two data samples $\mathbf{x}_i$ and $\mathbf{x}_j$. One of the commonly used correlation functions is Gaussian correlation function (Shi & Rasheed, 2010). Gaussian correlation function can be represented as follows:

$$R\big(\mathbf{x}_i, \mathbf{x}_j\big) = \exp\left[-\sum_{k=1}^{d} \theta_k \big|x_{ik} - x_{jk}\big|^2\right] \tag{3.4}$$

, where d is the dimension of the problem, x_{ik} and x_{jk} are the kth element in the samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ respectively, and θ_k is the Gaussian correlation function parameter. Finally, the original function $f(\boldsymbol{x})$ can then be approximated by kriging:

$$\hat{f}(\mathbf{x}) = \hat{\beta} + r^T(\mathbf{x}) R^{-1}(\mathbf{y} - \hat{\beta}\mathrm{I}) \tag{3.5}$$

, where $\hat{f}(\mathbf{x})$ is the approximated value given the input $\boldsymbol{x}$ and $\boldsymbol{y} = \{y_1, \dots, y_n\}$, $\hat{\beta}$ is the estimated parameter for β, $r^T(\mathbf{x}) = [R(\boldsymbol{x}, \boldsymbol{x}_1), \dots, R(\boldsymbol{x}, \boldsymbol{x}_n)]^T$, and I is a unit vector with a length of n. The parameters can then be obtained using least squares or maximum likelihood method (Jin, 2005).

3.3.2. Polynomial regression

PR is capable of approximating nonlinear functions by introducing different degrees of order to the linear regression. For instance, a second-order polynomial regression model can be represented as follows:

$$\hat{f}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i \mathbf{x}_i + \sum_{i=1}^{n} \beta_{ii} \mathbf{x}_i^2 + \sum_{i} \sum_{j} \beta_{ij} \mathbf{x}_i \mathbf{x}_j \tag{3.6}$$

, where the β terms are the coefficients. These coefficients can be computed using least squares methods. The number of coefficient terms in the polynomial regression model is equal to $n_{coef} = (d+1)(d+2)/2$, where d is the dimension of the input space (i.e., the number of input features). In addition, for a second-order polynomial, it is recommended to include $1.5n_{coef}$, $3n_{coef}$, and $4.5n_{coef}$ data samples to construct the polynomial regression models for problems with a dimension of 5-10, 10-20, and 20-30 input variables respectively (Jin *et al.*, 2001).

3.3.3. Radial basis function network

RBF network is a type of neural network, which consists of an input layer, hidden layer, and output layer. The input layer consists of input neurons each of which represents each data sample. Each of hidden neurons in the hidden layer is associated with the radial basis function. The number of hidden neurons in a RBF network can be as many as the number of data samples n. In the output layer, there exists an output neuron or multiple output neurons depending on the number of features in the problem. A RBF network can be written as follows:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} w_i \gamma_i \|\mathbf{x} - \mathbf{x}_i\| + w_0 \tag{3.7}$$

, where γ_i is the radial basis function for the *i*th hidden neuron, w_0 is the bias term, w_i is the weight coefficient associated with $\mathbf{x}_i$. Note that the term $\gamma_i\|\mathbf{x} - \mathbf{x}_i\|$ essentially computes the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$. In case the number of data samples is large, the number of hidden neurons also increases. This makes it difficult to implement on such a large dataset requiring more computational effort. Therefore one can utilize a generalized RBF network as follows:

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{J} w_j \gamma_j \|\mathbf{x} - c_j\| + w_0 \tag{3.8}$$

, where c_j is the center, which should be determined by the user. Then, it is a major task to determine how many number of centers and where to locate them. The most common RBF function is Gaussian kernel function (Jin, 2005), which can be represented as follows:

$$\gamma_j \|\mathbf{x} - c_j\| = exp\left(-\frac{\|\mathbf{x} - c_j\|}{2\sigma^2}\right) \tag{3.9}$$

Once the centers are determined, the bias and weight terms w_0 and w_j can be calculated by minimizing the sum of squares (i.e., linear least squares).

3.3.4. Generalized regression neural network

GRNN was first introduced by Specht (1991). In this research, GRNN will be used to construct a meta-model at each iteration of PSO in order to approximate the fitness function value of particles. GRNN has several advantages to be used as a meta-

modeling algorithm. GRNN is an instance-based learning algorithm, which can greatly reduce the meta-model training time, therefore, the overall computation cost of PSO as well. Its only one parameter, smoothing factor, is not very sensitive to its setting (Gheyas & Smith, 2010). In addition, due to the low sensitivity of the smoothing factor, the optimal selection of this parameter is not as much necessary as other algorithms including MLP and SVR. Some other advantages include its ability to avoid a local minima and over-fitting to the training data, and robustness against noise (Currit, 2002; Yagci *et al.*, 2005; Białobrzewski, 2008).

GRNN considers each training data as a cluster. Once it takes a new input data $\boldsymbol{x}$ for the prediction of the output value, it calculates the Euclidean distance between the input $\boldsymbol{x}$ and each training data $\boldsymbol{x}_i$. The distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$ is calculated as follows:

$$D_i^2(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} - \mathbf{x}_i)^{\mathrm{T}} \cdot (\mathbf{x} - \mathbf{x}_i) \tag{3.10}$$

, where each input data is defined as $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$. d is the number of features in the problem. Note that, in case of PSO, we define the dimension of the problem as D, which should not be confused when GRNN is used as the meta-modeling algorithm for PSO. GRNN calculates the predicted output given an input, $\widehat{f}(\boldsymbol{x})$, according to the equation below:

$$\widehat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} \frac{f(\mathbf{x}_i) e^{\left(-\frac{D_i^2}{2\sigma^2}\right)}}{\sum_{i=1}^{n} e^{\left(-\frac{D_i^2}{2\sigma^2}\right)}} \tag{3.11}$$

where n is the number of training data and σ is the smoothing factor. Note that the predicted output is a weighted average of the actual outputs of all training data where the weights are the Euclidean distance between $\boldsymbol{x}$ and each training data $\boldsymbol{x}_i$. As mentioned

above, the smoothing factor is not very sensitive and also is not the main goal of this research, thus it is set to $\sigma = 1$ unless specified otherwise.

3.4. Meta-modeling using generalized regression neural network and particle swarm optimization (MUGPSO)

PSO can be used to solve an optimization problem. We briefly introduce a variant PSO, called global PSO (GPSO), proposed by Shi & Eberhart (1998a). First, we denote the number of particles in a swarm as N_s and refer a particle to as $\mathbf{P}_l$ for $l = 1, \dots, N_s$. Each particle in a swarm is a candidate solution to the given optimization problem of dimension D. Let $\mathbf{P}_l^t = (p_{l1}^t, \dots, p_{lD}^t)$ denote the position vector of particle l at iteration t. Accordingly, a swarm with N_s number of particles is represented as $S^t = (\mathbf{P}_1^t, \dots, \mathbf{P}_{N_s}^t)$ at iteration t.

At each iteration, PSO keeps track of the local best and global best particles. $\mathbf{L}_l^t = (l_{l1}^t, \dots, l_{lD}^t)$ is defined as the local best solution obtained over iterations t for particle l. Therefore, we have N_s local best solutions for each particle l at iteration t. Similarly, the global best solution at iteration t is represented as $\mathbf{G}^t = (g_1^t, \dots, g_D^t)$.

Every particle in the swarm is moved by some portion of its local best and the global best solution at each iteration, so that the entire swarm can also move towards to the optimal solution. The particles are updated as follows:

$$\mathbf{V}_l^{t+1} = w\mathbf{V}_l^t + c_1 r_1(\mathbf{L}_l^t - \mathbf{P}_l^t) + c_2 r_2(\mathbf{G}^t - \mathbf{P}_l^t) \tag{3.12}$$

$$\mathbf{P}_l^{t+1} = \mathbf{P}_l^t + \mathbf{V}_l^{t+1} \tag{3.13}$$

where w is inertia weight, c_1 and c_2 are called acceleration coefficients for the local and global best solutions, and r_1 and r_2 are uniform random numbers distributed in [0,1]. The

particle update equation stochastically moves each particle around its local and global best solutions using the inertia weight and acceleration coefficients. The acceleration coefficients c_1 and c_2 are set to 2 in this research based on reported empirical studies (Clerc & Kennedy, 2002; Shi & Eberhart, 1998b). The inertia weight controls the extent to which the memory of the particle's previous velocity influences the new velocity. It was reported in Van den Bergh & Engelbrecht (2006) that velocities quickly explode to large values in the early iterations, especially for particles far from the local and global best. This allows particles to move beyond the boundaries of the search space, which results in divergent solutions. Inertia weight helps prevent particles to diverge by controlling the contribution of previous movement direction by allowing bigger movements at the beginning and smaller movements towards to the end of the run, that is the inertia weight decreases as the iteration proceeds (Van den Bergh & Engelbrecht, 2006). One can define the inertia weight as follows:

$$w^{t+1} = w_{start} - \frac{(w_{start} - w_{end})t}{t_{max}} \tag{3.14}$$

where t_{max} is the maximum number of iterations and w_{start} and w_{end} are the starting and end values of the inertia weight respectively. We set w_{start} and w_{end} set to 0.9 and 0.4 respectively based on previous research conducted in (Shi & Eberhart, 1998b).

Figure 3.2. describes the entire procedure of MUGPSO. In order to run a MUGPSO, one should initialize several parameters related to PSO mentioned in the previous section. These PSO parameters include the inertia weight, acceleration coefficients, maximum number of iterations, swarm size, etc. All the parameters used in this research are reported in Table 3.1. The selection scheme for particles to be re-

evaluated with the real fitness function is inspired by the global model included in TLSAPSO (Sun *et al.*, 2014).

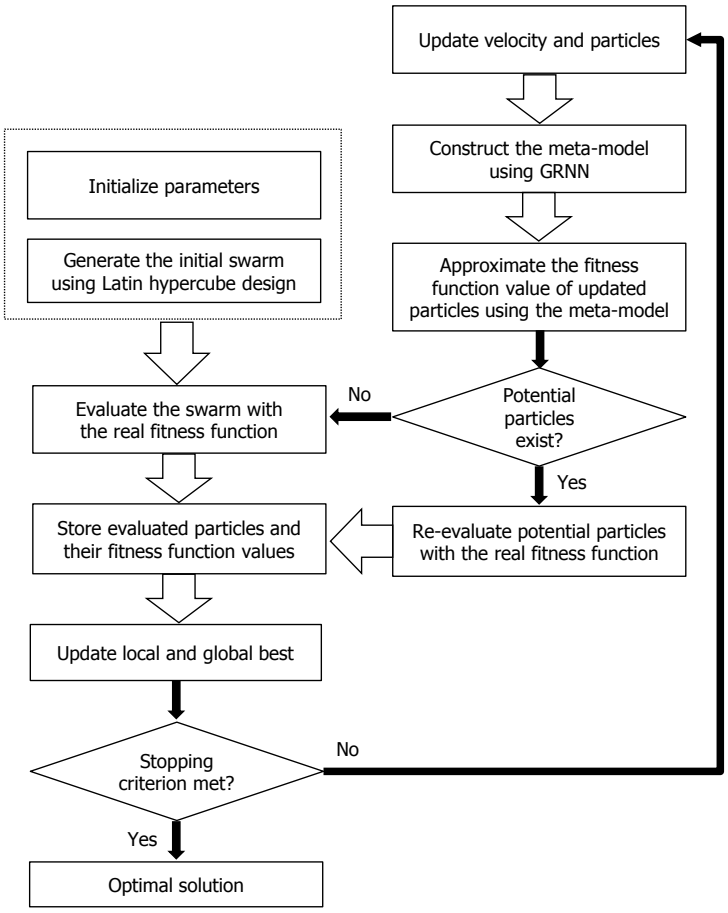


Figure 3.2. Schematic procedure of MUGPSO

Table 3.1. Parameter settings of GPSO and MUGPSO

Parameter	Value
Problem dimension (D)	30
Number of particles (N_s)	60
Acceleration coefficients (c_1, c_2)	2
Inertia weights (w_{start}, w_{end})	0.9 and 0.4
Maximum number of iteration (t_{max})	166 for GPSO MUGPSO adaptively sets for each benchmark problem
Maximum number of real fitness function evaluations	10,000
Storage threshold (δ)	0.001
Smoothing factor Maximum number of iteration (σ)	1

As shown in Figure 3.2., MUGPSO generates an initial swarm of particles by employing the Latin hypercube design method. Then, this initial swarm is evaluated with the real fitness function for its particles. The particles and their original fitness function values in the initial swarm are stored for constructing meta-models. The particles are updated using Equation (3.13).

Throughout the iterations, MUGPSO, similar to PSO, updates the velocity and particles. A meta-model is then constructed using GRNN on the closest data in the storage from the current updated particles. The meta-model approximates the fitness function on these updated particles. As mentioned in the previous section, PSO aims to reach the global optimum by moving its particles by some portion of its local best and the global best solution at each iteration. For this reason, MUGPSO re-evaluates particles whose approximated fitness function values are better than their local best with the real fitness function.

The particles re-evaluated with the real fitness function will be considered informative and stored for constructing meta-models in the later iterations. In the current version of MUGPSO, we define *a particle is informative for PSO if the portion of improvement obtained by re-evaluated real fitness function value is larger than the approximated value by the meta-model*. This can denote that we can expect to construct meta-models to approximate the real fitness function close enough using the information that have been stored. This approach is simple, however, and showed promising results in (Park & Lee, 2014; *Sun et al.*, 2014) since local and global best solutions are important for PSO to move toward the optimal solution region.

Obviously, as the iteration process proceeds, the number of particles and the fitness function values increases. Since MUGPSO uses the closest particle and its fitness function value to train the global meta-model at each iteration, the size of the particle storage affects the computational efficiency in calculating the distance from each particle to those in the storage. This is an important area to further research and improve the current version of MUGPSO.

Pseudo code: MUGPSO
1: t=0
2: Swarm initialization $S^t = (\mathbf{P}_1^t, \dots, \mathbf{P}_{N_s}^t)$ *with Latin hypercube design*
3: Evaluate S^t *with the real fitness function*
4: Define local best positions $\mathbf{L}_l^t = \mathbf{P}_l^t$
5: Define global best position $\mathbf{G}^t = min(fitness(\mathbf{P}_l^t))$
6: while (stopping criteria not met)
7: Update velocities using Equation 1
8: Update particles using Equation 2
9: Construct a meta-model using GRNN with the closest data samples from the storage
10: Estimate fitness function values of particles using the meta-model
11: if at least one such particle l exists that $f(\mathbf{L}_l^t) < \hat{f}(\mathbf{P}_l^{t+1})$
12: for each particle l
13: if $f(\mathbf{L}_l^t) < \hat{f}(\mathbf{P}_l^{t+1})$
14: Evaluate $\mathbf{P}_l^{t+1}$ *with the real fitness function*
15: if $\left|(\hat{f}(\mathbf{P}_l^{t+1}) - f(\mathbf{P}_l^{t+1}))/f(\mathbf{P}_l^{t+1})\right| > \delta$
16: Store $\mathbf{P}_l^{t+1}$ *and* $f(\mathbf{P}_l^{t+1})$
17: end if
18: end if
19: end for
20: else
21: Evaluate the whole swarm with the real fitness function
22: Store the whole swarm and fitness function values
23: end if
24: Update local best positions and global best position
25: t=t+1
26: end while

Figure 3.3. A pseudo code for MUGPSO

The above steps are repeated until a stopping criterion is met. MUGPSO stops when the maximum number of iteration or the maximum number of real fitness function evaluations is reached. A pseudo code of MUGPSO is shown in Figure 3.3.

3.5. Experimental results

We examine the performance of MUGPSO on ten benchmark problems from Suganthan *et al.* (2005). Comparisons with other meta-modeling techniques reported in Sun *et al.* (2014) will also be discussed with these ten benchmark problems. They cover various characteristics of optimization problems as indicated for each problem below. The ten benchmark problems and their characteristics are listed as follows:

(1) Shifted sphere

$$\mathrm{F1}(\mathbf{x}) = \sum_{j=1}^{30} z_j^2 - 450 \tag{3.15}$$

Unimodal
$x_j \in [-100, 100]$
$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o}$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F1}(\mathbf{x}^*) = -450$
$t_{max} = 400$

(2) Shifted Schwefels problem 1.2

$$\mathrm{F2}(\mathbf{x}) = \sum_{j=1}^{30} \left(\sum_{k=1}^{j} z_k \right)^2 - 450 \tag{3.16}$$

Unimodal
$x_j \in [-100, 100]$
$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o}$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F2}(\mathbf{x}^*) = -450$
$t_{max} = 700$

(3) Shifted rotated high conditioned elliptic

$$\mathrm{F3}(\mathbf{x}) = \sum_{j=1}^{30} (10^6)^{\frac{j-1}{30-1}} z_j^2 - 450 \tag{3.17}$$

Unimodal
$x_j \in [-100, 100]$
$\boldsymbol{z} = (\mathbf{x} - \boldsymbol{o})\boldsymbol{M}$
$\boldsymbol{M}$: orthogonal matrix
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F3}(\mathbf{x}^*) = -450$
$t_{max} = 800$

(4) Noise is added to F2

$$\mathrm{F4}(\mathbf{x}) = \left(\sum_{j=1}^{30} \left(\sum_{k=1}^{j} z_k \right)^2 \right) (1 + 0.4|N(0,1)|) - 450 \tag{3.18}$$

Unimodal
Gaussian noise added from $\mathrm{N}(0,1)$
$x_j \in [-100, 100]$
$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o}$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F4}(\mathbf{x}^*) = -450$
$t_{max} = 800$

(5) Schwefels problem 2.6 with global optimum on bounds

$$\mathrm{F5}(\boldsymbol{x}) = \max\{|\boldsymbol{A}_j \mathbf{x} - \boldsymbol{B}_j|\} - 310 \tag{3.19}$$

Unimodal
$\boldsymbol{A}$: 30x30 matrix, a_{ij} is a uniform random number from $[-500, 500]$
$\boldsymbol{B}_i = \boldsymbol{A}_i \times \boldsymbol{o}$, each o_i is a random number from $[-100, 100]$
$x_j \in [-100, 100]$
$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o}$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F5}(\mathbf{x}^*) = -310$
$t_{max} = 800$

(6) Shifted Rosenbrocks

$$\mathrm{F6}(\mathbf{x}) = \sum_{j=1}^{30-1} \left(100\left(z_j^2 - z_{j+1}\right)^2 + \left(z_j - 1\right)^2 \right) + 390 \tag{3.20}$$

Multimodal

A very narrow valley around local optima
$x_j \in [-100, 100]$
$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o} + 1$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F6}(\mathbf{x}^*) = 390$
$t_{max} = 400$

(7) Shifted rotated Griewanks without bounds

$$\mathrm{F7}(\mathbf{x}) = \sum_{j=1}^{30} \frac{z_j^2}{4000} - \prod_{j=1}^{30} \cos\left(\frac{z_j}{\sqrt{j}}\right) + 1 - 180 \tag{3.21}$$

Multimodal
Initial swarm from $x_j \in [0, 600]$
The global optimum locates outside of the range
$\boldsymbol{z} = (\mathbf{x} - \boldsymbol{o})\boldsymbol{M}$
$\boldsymbol{M}$: linear transformation matrix
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F7}(\mathbf{x}^*) = -180$
$t_{max} = 300$

(8) Shifted rotated Ackleys with global optimum on bounds

$$\mathrm{F8}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{j=1}^{30} z_j^2}\right) - \exp\left(\frac{1}{30}\sum_{j=1}^{30}\cos(2\pi z_j)\right) + 20 - 140 \tag{3.22}$$

Multimodal
Global optimum on the boundary
Narrow region with many local optima around the global optimum
$x_j \in [-32, 32]$
$\boldsymbol{z} = (\mathbf{x} - \boldsymbol{o})\boldsymbol{M}$
$\boldsymbol{M}$: linear transformation matrix
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F8}(\mathbf{x}^*) = -140$
$t_{max} = 400$

(9) Shifted Rastrigins

$$\mathrm{F9}(\mathbf{x}) = \sum_{j=1}^{30}\left(z_j^2 - 10\cos(2\pi z_j) + 10\right) - 330 \tag{3.23}$$

Multimodal
Many local optima
$x_j \in [-5, 5]$

$\boldsymbol{z} = \mathbf{x} - \boldsymbol{o}$
$\mathbf{x}^* = \boldsymbol{o}$
$\mathrm{F9}(\mathbf{x}^*) = -330$
$t_{max} = 500$

(10) Shifted rotated Rastrigins

$$\mathrm{F10}(\mathbf{x}) = \sum_{j=1}^{30} \left(z_j^2 - 10\cos\left(2\pi z_j\right) + 10\right) - 330 \tag{3.24}$$

Multimodal
Many local optima
$x_j \in [-5, 5]$
$\boldsymbol{z} = (\mathbf{x} - \boldsymbol{o})\boldsymbol{M}$
$\boldsymbol{M}$: linear transformation matrix
$\mathbf{x}^* = \mathbf{o}$
$\mathrm{F10}(\mathbf{x}^*) = -330$
$t_{max} = 1200$

Before we compare MUGPSO with other meta-modeling algorithms, each benchmark function is tested with GPSO (global PSO) and MUGPSO. GPSO is a variant of PSO and does not approximate the fitness function using meta-models. Therefore, every candidate solution throughout the entire run has to be evaluated with the real fitness function. Table 3.1. shows the parameter settings for GPSO and MUGPSO. In order to compare MUGPSO to others, most parameters are consistently set the same as in Sun *et al.* (2014).

The maximum number of iterations for GPSO is set to 166, which allows 9,960 real fitness function evaluations since the number of particles is 60. For MUGPSO, the maximum number of iterations t_{max} is adaptively set according to the maximum number of real fitness function evaluations allowed. The maximum number of real fitness function is set to 10,000, which is equivalent to the settings in Sun *et al.* (2014) for a comparison purpose. For instance, in order to set t_{max} for $\mathrm{F1}(\mathbf{x})$, we ran MUGPSO

several times to see what iteration MUGPSO reaches around 10,000 real fitness function evaluations. MUGPSO roughly reached 400 iterations when the maximum number of real fitness function evaluations is used. Note that, in most of cases for every benchmark functions, MUGPSO stopped using less than 10,000 real fitness function evaluations (see Table 3.2.).

Table 3.2. Statistical results of solutions obtained by GPSO and MUGPSO with a limit of 10,000 real fitness function evaluations

	Algorithms	Global optimum	Best	Worst	Mean	SD	Number of Evaluations
F1	GPSO	-4.50e+02	-3.88e+02	-1.28e+02	-2.49e+02	9.55e+01	9960
	MUGPSO		-4.29e+02	-3.19e+02	-4.03e+02	3.27e+01	10016
F2	GPSO	-4.50e+02	6.44e+03	1.76e+04	1.34e+04	3.63e+03	9960
	MUGPSO		2.21e+03	1.59e+04	8.70e+03	4.35e+03	9863
F3	GPSO	-4.50e+02	3.65e+07	1.68e+08	7.39e+07	3.87e+07	9960
	MUGPSO		2.16e+07	5.87e+07	3.39e+07	1.28e+07	9649
F4	GPSO	-4.50e+02	8.30e+03	4.58e+04	2.92e+04	1.04e+04	9960
	MUGPSO		9.66e+03	3.20e+04	2.17e+04	8.39e+03	9026
F5	GPSO	-3.10e+02	6.00e+03	8.13e+03	7.32e+03	7.51e+02	9960
	MUGPSO		5.01e+03	6.75e+03	5.60e+03	5.17e+02	9441
F6	GPSO	3.90e+02	5.86e+05	3.33e+06	1.87e+06	1.11e+06	9960
	MUGPSO		1.45e+04	1.05e+05	0.06e+06	3.49e+04	10016
F7	GPSO	-1.80e+02	-1.73e+02	-1.51e+02	-1.63e+02	6.47e+00	9960
	MUGPSO		-1.76e+02	-1.67e+02	-1.71e+02	3.21e+00	10002
F8	GPSO	-1.40e+02	-1.18e+02	-1.18e+02	-1.18e+02	0.04e+00	9960
	MUGPSO		-1.19e+02	-1.18e+02	-1.18e+02	0.07e+00	8380
F9	GPSO	-3.30e+02	-2.56e+02	-1.51e+02	-2.14e+02	3.30e+01	9960
	MUGPSO		-2.87e+02	-1.95e+02	-2.51e+02	3.03e+01	8189
F10	GPSO	-3.30e+02	-1.37e+02	-8.18e+01	-1.07e+02	1.59e+01	9960
	MUGPSO		-2.11e+02	-8.40e+01	-1.37e+02	4.54e+01	9692

Each algorithm is independently run for ten times. The comparative results report the mean, standard deviation, best, and worst solution achieved from the ten repetitions for each benchmark function. In addition, the mean number of real fitness function evaluations is also reported. These results are shown in Table 3.2. We report the

statistical results obtained by GPSO and MUGPSO when the number of real fitness function evaluations is limited to 20,000 in Appendix I.

Given that the number of real fitness function evaluations is limited to 10,000, one can claim (from Table 3.2.) that MUGPSO provides a quality optimal solution to all benchmark problems when compared to the results of GPSO. For benchmark problems F1, F3, F5, F6, F7, and F9, MUGPSO seems to outperform GPSO in terms of all the statistics used, mean, standard deviation, best, and worst solutions. It appears that MUGPSO provides a slight higher standard deviation and best solutions for F2, F4, and F10 respectively. For F8, MUGPSO only seems to outperform GPSO in terms of the best solution and the results do not seem to differ significantly. Note that F8 is known to have the global optimum on the boundary in a narrow region. Both GPSO and MUGPSO cannot seem to escape from local optimum.

Table 3.3. *t*-test results for the results from the ten repetitions of GPSO and MUGPSO

Benchmark Problem	(GPSO > MUGPSO) p-value
F1	2.7088e-04*
F2	0.0091*
F3	0.005*
F4	0.0468*
F5	1.1067e-05*
F6	3.0488e-04*
F7	0.0019*
F8	0.9118
F9	0.0089*
F10	0.0366*

Additionally, we perform *t*-tests to identify the difference between GPSO and MUGPSO using the results achieved from the ten repetitions. The alternative hypothesis is that MUGPSO provides better solutions than GPSO when the number of real fitness

function evaluations is limited to 10,000. The significance level is set to $\alpha = 5\%$ and variance is assumed to be unequal. Table 3 shows the *t*-test results and they suggest that, except for F8, MUGPSO is likely to provide better solutions than GPSO, while number of real fitness function evaluations is limited.

Figures 3.4. through 3.7. show the convergence profile of GPSO and MUGPSO throughout the iterations. The mean fitness values displayed on the y-axis are calculated from the results obtained from the ten repetitions. The left hand figures are scaled to show all the optimal solution that GPSO and MUGPSO have visited at each iteration for each benchmark problem. Whereas in the right hand figures, they are scaled to zoom in so that the final solutions achieved by GPSO and MUGPSO can be shown better for comparison purposes. We have confirmed from the *t*-tests that MUGPSO is likely to obtain better solutions when the number of real fitness function evaluations is limited. We show these convergence profile figures for two of unimodal and multimodal problems that MUGPSO seems to excel more significant than in other problems, which are F1, F5, F6 and F7. The rest of convergence profile figures are in Appendix J.

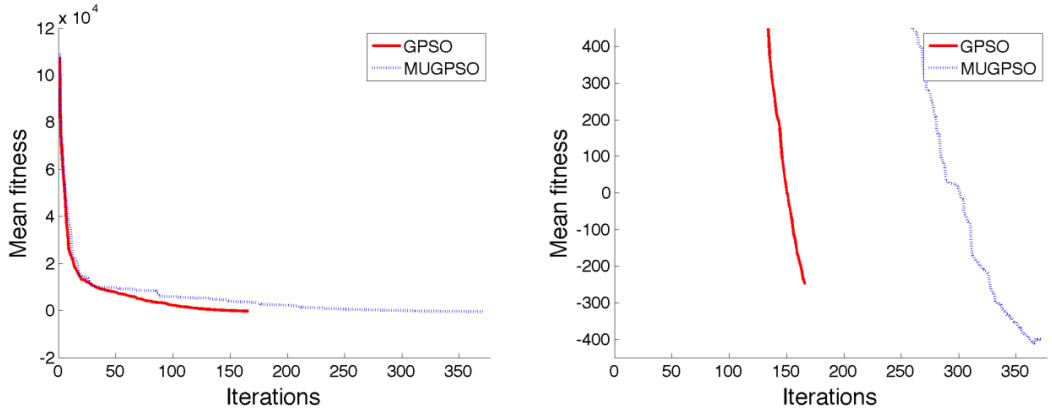


Figure 3.4. Convergence profile of GPSO and MUGPSO on F1. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right)

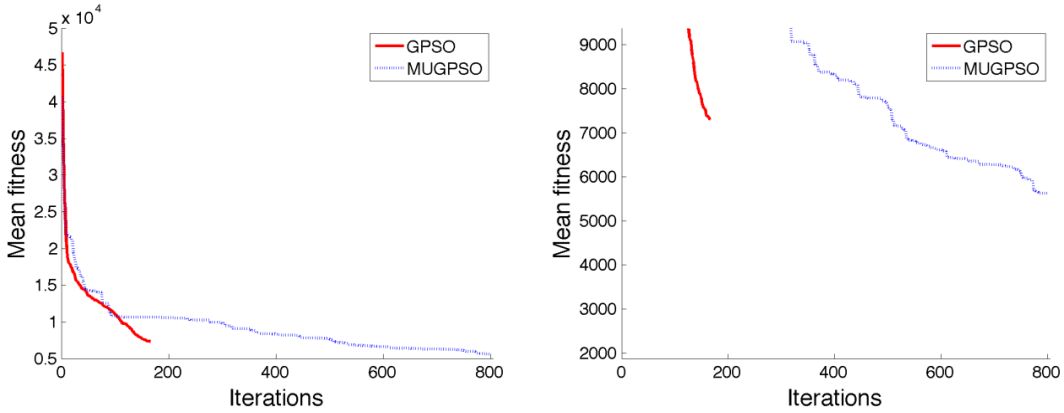


Figure 3.5. Convergence profile of GPSO and MUGPSO on F5. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right)

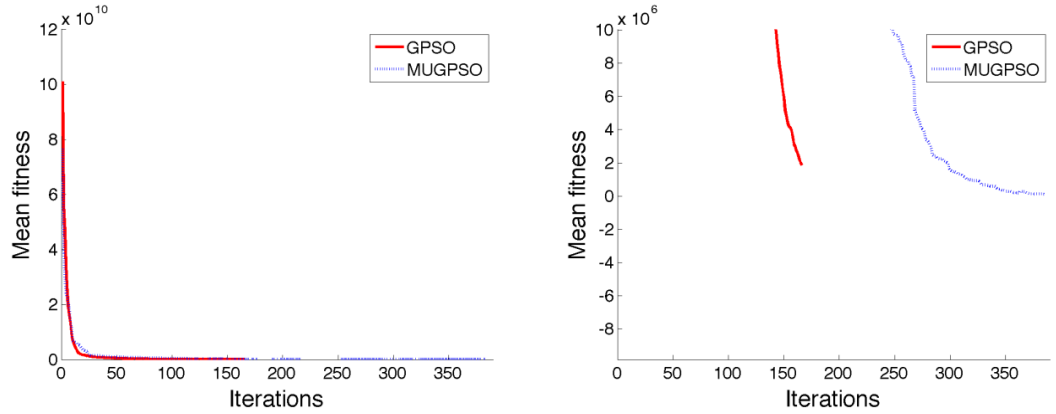


Figure 3.6. Convergence profile of GPSO and MUGPSO on F6. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right)

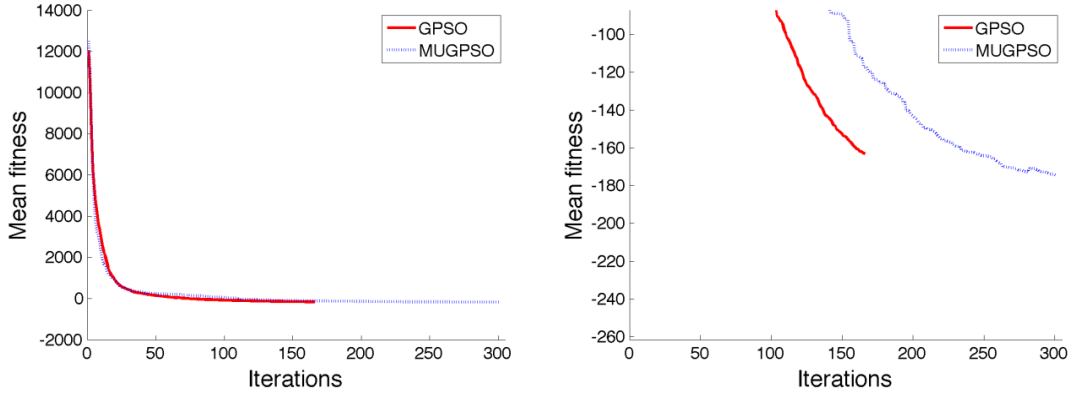


Figure 3.7. Convergence profile of GPSO and MUGPSO on F7. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right)

In addition to the above comparisons between GPSO and MUGPSO, we consider other meta-modeling techniques tested on the ten benchmark functions in the literature. We adopt the results reported in Sun *et al.* (2014) where a meta-modeling technique called two-layer surrogate-assisted particle swarm optimization (TLSAPSO) is proposed and compared with the results from other research. Their comparative results include the statistics of SVR-DE and SVC-DE (Lu *et al.*, 2011), FESPSO (Sun *et al.*, 2013), and TLSAPSO (Sun *et al.*, 2014) Table 3.4. describes the results of each of the algorithms on the ten benchmark functions.

Table 3.4. Results of MUGPSO, SVR-DE, SVC-DE, FESPSO, and TLSAPSO on the ten benchmark functions

	Algorithms	Global optimum	Best	Worst	Mean	SD
F1	MUGPSO	-4.50e+02	-4.29e+02	-3.19e+02	-4.03e+02	3.27e+01
	SVR-DE		0.46e+00	0.86e+00	0.63e+00	0.09e+00
	SVC-DE		0.06e+00	0.22e+00	0.11e+00	0.04e+00
	FESPSO		4.78e+02	5.12e+03	2.40e+03	1.79e+03
	TLSAPSO		**-4.50e+02**	**-4.49e+02**	**-4.50e+02**	**3.90e-03**
F2	MUGPSO	-4.50e+02	2.21e+03	1.59e+04	8.70e+03	4.35e+03
	SVR-DE		6.72e+03	2.45e+04	1.64e+04	4.87e+03
	SVC-DE		1.72e+03	7.12e+03	3.54e+03	**1.33e+03**
	FESPSO		**9.85e+02**	**6.01e+03**	**3.08e+03**	1.71e+03
	TLSAPSO		3.57e+03	7.99e+03	5.75e+03	1.44e+03
F3	MUGPSO	-4.50e+02	2.16e+07	5.87e+07	3.39e+07	1.28e+07
	SVR-DE		5.82e+07	1.68e+08	1.10e+08	2.75e+07
	SVC-DE		7.38e+06	3.42e+07	1.80e+07	**5.75e+06**
	FESPSO		8.33e+06	2.25e+08	5.59e+07	6.74e+07
	TLSAPSO		**5.64e+06**	**3.01e+07**	**1.57e+07**	7.72e+06
F4	MUGPSO	-4.50e+02	9.66e+03	3.20e+04	2.17e+04	8.39e+03
	SVR-DE		1.20e+04	3.83e+04	2.70e+04	7.06e+03
	SVC-DE		**3.67e+03**	**1.27e+04**	**7.71e+03**	**2.77e+03**
	FESPSO		9.05e+03	2.91e+04	1.85e+04	7.24e+03
	TLSAPSO		1.04e+04	2.55e+04	1.75e+04	3.84e+03
F5	MUGPSO	-3.10e+02	5.01e+03	6.75e+03	5.60e+03	**5.17e+02**
	SVR-DE		**7.30e+02**	3.28e+03	**2.24e+03**	5.69e+02
	SVC-DE		1.49e+03	**3.27e+03**	2.39e+03	5.71e+02
	FESPSO		7.95e+03	1.67e+04	1.20e+04	2.84e+03
	TLSAPSO		5.25e+03	1.54e+04	1.01e+04	2.93e+03
F6	MUGPSO	3.90e+02	1.45e+04	1.05e+05	0.06e+06	3.49e+04

	SVR-DE		5.11e+06	7.16e+07	2.32e+07	1.43e+07
	SVC-DE		1.08e+03	1.04e+04	2.54e+03	3.11e+03
	FESPSO		3.72e+06	1.47e+09	5.32e+08	4.74e+08
	TLSAPSO		**5.82e+02**	**6.42e+03**	**1.57e+03**	**1.76e+03**
F7	MUGPSO	-1.80e+02	-1.76e+02	-1.67e+02	-1.71e+02	3.21e+00
	SVR-DE		1.02e+00	1.12e+00	1.06e+00	2.35e-02
	SVC-DE		1.17e-01	4.40e-03	4.03e-02	3.15e-02
	FESPSO		-1.79e+02	-1.74e+02	-1.77e+02	1.59e+00
	TLSAPSO		**-1.79e+02**	**-1.75e+02**	**-1.78e+02**	**1.03e+00**
F8	MUGPSO	-1.40e+02	**-1.19e+02**	-1.18e+02	-1.18e+02	0.07e+00
	SVR-DE		2.09e+01	2.12e+01	2.11e+01	6.39e-02
	SVC-DE		2.09e+01	2.12e+01	2.08e+01	6.61e-02
	FESPSO		**-1.19e+02**	**-1.19e+02**	**-1.19e+02**	**1.43e-01**
	TLSAPSO		**-1.19e+02**	**1.19e+02**	**-1.19e+02**	4.93e-02
F9	MUGPSO	-3.30e+02	**-2.87e+02**	-1.95e+02	**-2.51e+02**	3.03e+01
	SVR-DE		1.79e+02	2.17e+02	2.01e+02	**1.14e+01**
	SVC-DE		1.84e+02	2.27e+02	2.09e+02	1.31e+01
	FESPSO		-2.82e+02	-1.95e+02	-2.37e+02	2.93e+01
	TLSAPSO		-2.73e+02	**-2.00e+02**	-2.29e+02	2.39e+01
F10	MUGPSO	-3.30e+02	-2.11e+02	-8.40e+01	-1.37e+02	4.54e+01
	SVR-DE		1.80e+02	2.34e+02	2.15e+02	**1.29e+01**
	SVC-DE		1.93e+02	2.38e+02	2.15e+02	1.37e+01
	FESPSO		-2.11e+02	-6.05e+01	-1.57e+02	5.06e+01
	TLSAPSO		**-2.60e+02**	**-1.14e+02**	**-1.91e+02**	4.96e+01

Overall, MUGPSO outperforms others on F9 in terms of the mean and best solutions from the ten repetitions. As for other benchmark functions, TLSAPSO, SVR-DE, and SVC-DE seem to provide better solutions. Comparing the results further, MUGPSO does not provide the worst solutions for any of the ten benchmark functions and yet the results obtained by MUGPSO are comparable to the best ones for every benchmark function. One should note that the current version of MUGPSO is designed to be as simple as possible, in order to identify the applicability of GRNN as a meta-modeling algorithm. The computational cost of MUGPSO in constructing meta-models throughout the iterations is likely to be less than the other meta-modeling algorithms. In addition, RBF network, SVM, and SVR are employed respectively in TLSAPSO, SVR-

DE, and SCV-DE. These algorithms require more parameters and iterations to construct a meta-model. The effort of optimizing parameters with GRNN is expected to be smaller than others, since its only one parameter is known insensitive. In the current version of MUGPSO, pre-defined smoothing factor is used for all the meta-models constructed throughout the iterations and the results are satisfactory.

In order to compare the computational complexity in constructing the meta-models, we analyze the complexity of MUGPSO and compared it to that of TLSAPSO (Sun *et al.*, 2014), which provided superior results on eight benchmark problems considered. MUGPSO evaluates all the particles at the first iteration with the real fitness function, which takes $O(N_s D)$. The initial particles and their real fitness function values are then stored for constructing meta-models. TLSAPSO performs the same task at the first iteration.

Starting from the second iteration, a meta-model is constructed using the closest data samples in the storage to the particles in the current swarm. The distance calculation for one particle requires $O(D)$. It requires $O(N_{st} D)$ to find the closest one in the storage, where N_{st} is the number of data samples in the storage. Therefore, we have $O(N_s N_{st} D)$ as the total complexity in calculating distances and finding the closest data samples in the storage for all the particles in a swarm. Once the closest data samples are obtained, GRNN can construct a meta-model and approximate the fitness function value for each particle. For GRNN to construct a meta-model, summations and multiplications in Equation (3.11) are required, which results in a linear time of $O(N_s)$. Finally, the computational complexity of MUGPSO in constructing meta-models is $O(N_s N_{st} D + N_s)$.

Each particle is approximated using the meta-model, particles whose approximation is better than its local best are re-evaluated with the real fitness function. These particles are selectively stored and its corresponding space complexity is $O(\alpha D)$, $0 < \alpha \leq N_s$, where α is the number of informative particles (i.e., the meta-model provides better fitness function values than the previous local best.). However, in the worst case, MUGPSO re-evaluates all the particles with the real fitness function, which makes the complexity $O(N_s D)$ in storing. Finally, the overall time complexity of MUGPSO in constructing meta-models is $O(N_s N_{st} D + N_s + N_s D)$ which is equivalent to $O(N_s(1 + D + N_{st} D))$.

Similarly, TLSAPSO starts constructing meta-models from the second iteration. TLSAPSO maintains two types of meta-models called the global and local throughout the run. The global model represents an entire swarm whereas the local models are constructed for each particle. The global model uses the closest data samples from the global database to construct a meta-model using RBF networks. Similar to MUGPSO, this process requires the distance calculation to find the closest one in the storage which results in a complexity of $O(N_s N_{gdb} D)$, where N_{gdb} is the number of data samples in the global database. The average computational complexity of constructing RBF networks is $O(N_s \log N_s + N_s D \log N_s)$ (Oyang *et al.*, 2005). The local model is constructed for each particle when the number of data samples in each local database exceeds a threshold. This requires $O(N_s^* \log N_s^* + N_s^* D \log N_s^*)$ for each particle where N_s^* is an arbitrary number of data samples that the local model use to construct the meta-model. The number of data samples is different for each local model for each particle. In the worst case, TLSAPSO constructs local meta-model for every particle in the swarm, which

makes a total of $O(N_s(N_s^* \log N_s^* + N_s^* D \log N_s^*))$ for constructing local meta-models. Similar to MUGPSO, TLSAPSO re-evaluates particles with the real fitness function if the approximation is better than the previous local best. In the worst case, the whole swarm is re-evaluated with the real fitness function. This requires the same complexity of $O(N_s D)$. The overall time complexity of TSLAPSO in constructing meta-models is then $O\left(N_s N_{gdb} D + N_s \log N_s + N_s D \log N_s + N_s(N_s^* \log N_s^* + N_s^* D \log N_s^*)\right)$.

3.6. Summary

A meta-modeling approach named MUGPSO is proposed. The main objective is to identify the capability of GRNN as a meta-modeling algorithm. For that purpose, the current version of MUGPSO employs the most basic meta-modeling construction and update scheme, where only one meta-model is maintained at each iteration. Regarding advantages of GRNN, the model construction does not require iterations. This can result in less computational costs in constructing meta-models if the size of storage is properly managed. Its only one parameter, called smoothing factor, is known insensitive, which requires less effort in optimizing the parameter.

The results obtained by MUGPSO and GPSO are compared. Given the limitation on the number of real fitness function evaluations, MUGPSO provide better results than GPSO for every benchmark function considered except for a benchmark problem F8, where both seem to struggle to escape from local minimum. Also, compared with several other meta-modeling approaches proposed in the literature (i.e., SVR-DE, SVC-DE, FESPSO, and TLSAPSO), we have identified that MUGPSO can support PSO in finding comparable optimal solutions. For F9 where the function characteristic is multimodal

with many local optimum, MUGPSO achieved the best results in terms of the mean and best solutions. Additionally, we compare the computational cost in constructing meta-models for MUGPSO and TLSAPSO and show how much computational cost can be reduced.

CHAPTER 4 META^2 PREDICTION MODELING FRAMEWORK

4.1. Overview of Meta^2

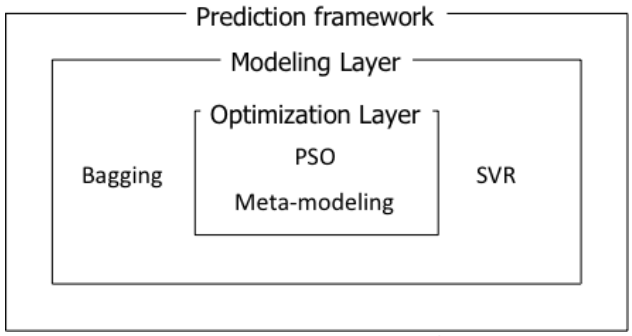


Figure 4.1. Illustration of the Meta^2 framework

Meta^2 consists of a modeling layer and optimization layer as shown in Figure 4.1. In the modeling layer, a prediction model is constructed using bagging with SVR as the base learning algorithm. The optimization layer aims to select the hyper-parameters for the bagging model in the modeling layer. The optimization problem is to select a set of hyper-parameters for SVRs that maximizes the prediction accuracy (i.e., minimizing the prediction error) given a number of SVRs in a bagging model. A bagging model uses the same hyper-parameters for its SVRs. PSO solves the optimization problem with meta-modeling. The fitness function is the prediction accuracy of a bagging model whereas each candidate solution represents a set of hyper-parameters. Meta-models are constructed by GRNN and approximate the fitness function of PSO (i.e., MUGPSO) in

order to reduce the number of bagging model constructions for candidate solutions. Figure 4.2. describes the overall procedure of Meta^2.

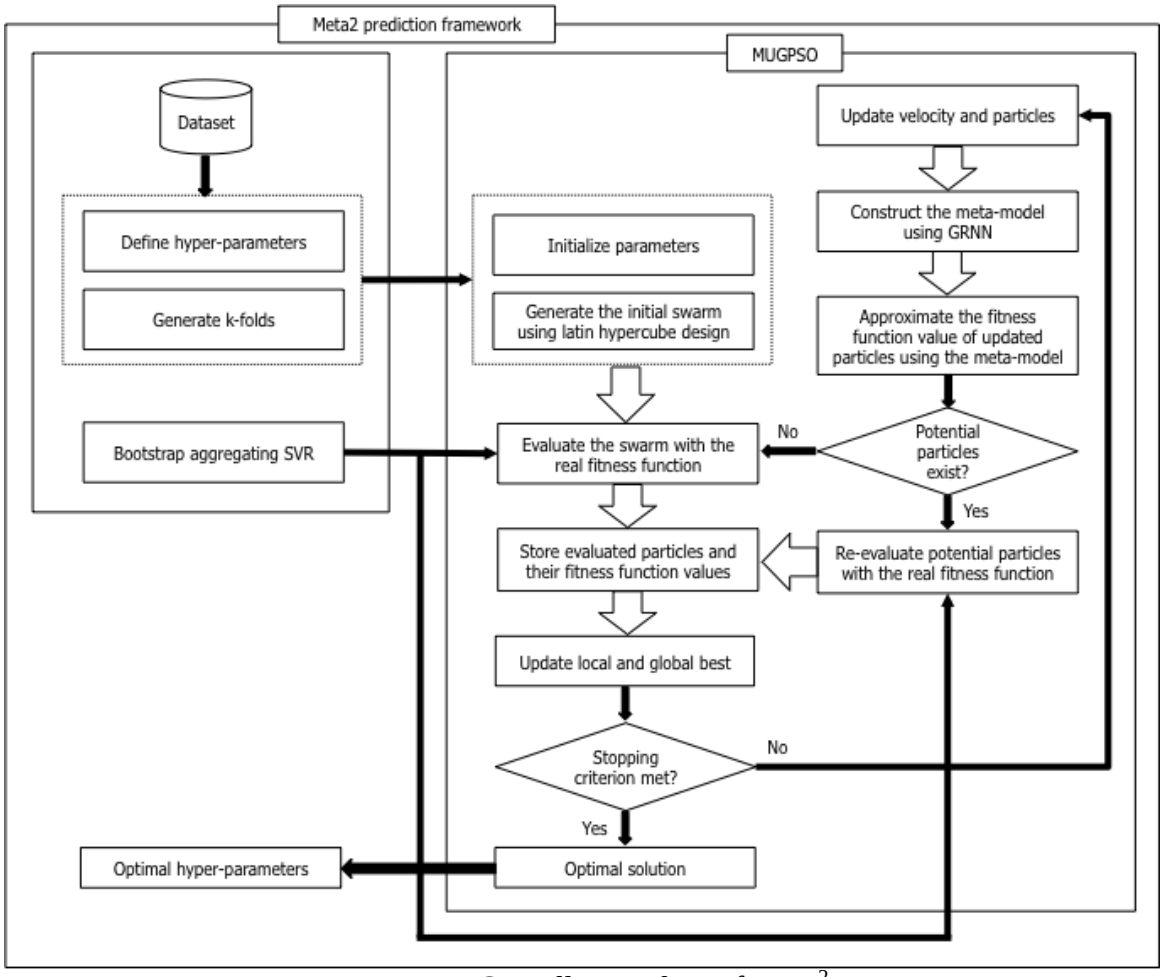


Figure 4.2. Overall procedure of Meta^2

4.2. Problem formulation for Meta^2

We mentioned in Section 2.2.2. that the main focus of Meta^2 is on the optimal hyper-parameter selection of SVRs in a bagging model. As for bagging, the size of each bootstrapped dataset is the same as the original dataset. The mean aggregating function is used in this research. The number of learning algorithms is a very important parameter to bagging in determining the prediction accuracy. However, as the number of learning

algorithms increases, the computational cost also dramatically increases. In Meta^2, the number of learning algorithms is a user input. SVR is chosen a default as the learning machine. Given the number of SVRs, Meta^2 applies the same hyper-parameters to all the SVRs in a bagging. The hyper-parameters selected by Meta^2 are the one that maximizes the prediction accuracy. We consider the RBF kernel function for bagging SVR since it has been successful in various applications for a single SVM and SVR (Chen, 2007; Lin *et al.*, 2008; Lins *et al.*, 2012).

Therefore, the hyper-parameters selection for bagging SVR is now formulated into a three dimensional optimization problem. The penalty coefficient C, ε precision, and γ for the RBF kernel function correspond to each of the three continuous-valued decision variables. In the context of MUGPSO, a swarm **S** consists of N_s number of three dimensional particles $\mathbf{P}_l$, each of which corresponds to a candidate solution of (C, ε, γ). We assign the decision space on each of these three variables as shown in Table 4.1.

Table 4.1. Decision boundary for hyper-parameters

	Decision boundary	Grid search
C	$[2^{-3}, 2^{15}]$	$[2^{-3}, 2^{-1}, 2^{1}, \dots, 2^{15}]$
ε	$[2^{-8}, 2^{1}]$	$[2^{-8}, 2^{-7}, 2^{6}, \dots, 2^{1}]$
γ	$[2^{-15}, 2^{3}]$	$[2^{-15}, 2^{-13}, 2^{-11}, \dots, 2^{3}]$

The optimal hyper-parameters will be found within these decision spaces. In addition, to compare the performance of Meta^2, we will also report the results of grid search. The grid search is designed as appeared in Table 4.1. Within each decision space for each hyper-parameter, the entire space is divided into ten points. This leaves 1,000 different combinations of hyper-parameters to construct bagging SVR models with. Note

that it may happen when the optimal hyper-parameters locate between two of these ten points, in which case the grid search will not be able to find the optimal solution. This decision space is set up based on previous research works (Fan et al., 2005; Moser & Serpico, 2009; Kavaklioglu, 2011).

The fitness function (i.e., objective function) is measured by the mean square error from a K-fold cross validation to avoid over-fitting. The entire dataset is randomly split into K folds. One fold is held out for testing, while the rest is used as a training dataset to construct a bagging SVR model. Prediction values for the testing dataset are calculated using the constructed bagging SVR model. Then, the mean square error for the kth fold is measured and the fitness for a particle is as follows:

$$\mathrm{MSE}_k = \frac{1}{n_k}\sum_{i=1}^{n_k}\left(\hat{f}_{bag}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2 \tag{4.1}$$

$$\mathrm{Fitness}(\mathbf{P}_l) = \mathrm{MSE} = \frac{\sum \mathrm{MSE}_k}{k} \tag{4.2}$$

where n_k is the number of testing data samples in the *k*th fold. This process continues for the rest of the folds. Finally, the mean square errors for every kth fold test datasets are averaged and represented as the fitness function value. Note that this fitness function can be replaced by other loss functions, depending on the requirement of the given problem. For instance, 0-1 loss function may be a better choice if this framework is applied to a classification problem.

4.3. Experimental results

We evaluate Meta^2 on several noisy datasets. Three datasets are artificially generated from using the sinc function. Also, the results on a resistance spot welding

(RSW) quality dataset are discussed. We use LIBSVM (Chang & Lin, 2011) for the implementation of ε-SVR. For PSO, a variant PSO, called global PSO (GPSO), is used (Shi & Eberhart, 1998a). The inertia weight for velocity vectors is employed with an initial value of 0.9 and end value of 0.4. The number of swarm is set to 30, which is 10×3, the number of decision variables. Since the stochastic nature of PSO, both GPSO and our proposed framework are repeatedly for ten times with the same setting. As for the number of SVRs in a bagging model, we consider 5, 10, … , 50 to identify the optimal number of SVRs.

4.3.1. Artificial datasets

Three artificial datasets are generated using a univariate sinc function. Different levels of Gaussian noise are added. The three sinc functions are generated as follows:

$$y = f(x) = \frac{\alpha \sin(x)}{x} \tag{4.3}$$

where $\alpha = 1, 10, 0.1$. A predefined number of x values are samples on uniformly spaced grid $x \in [-10,10]$. Gaussian noise δ with zero mean and standard deviation σ is then added to each of the three functions such that $y = f(x) + \delta$. Table 4.2. shows characteristics of the three datasets. Figure 4.3 through 4.5 illustrate these three datasets.

Table 4.2. Noisy artificial dataset characteristics

Dataset	α	Noise level σ	Sample space	Number of x values
1	1	0.2	$x \in [-10,10]$	37
2	10	2		
3	0.1	0.02		

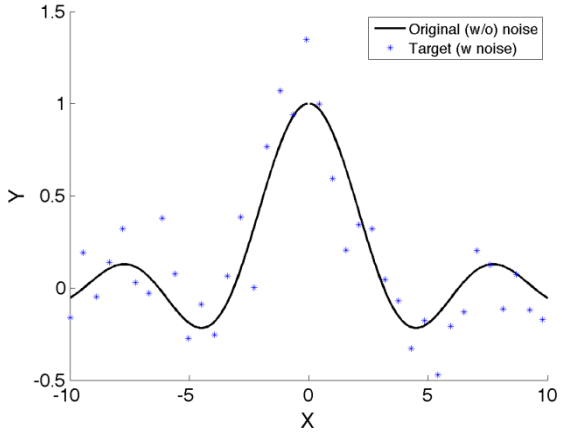


Figure 4.3. Visualization of dataset1

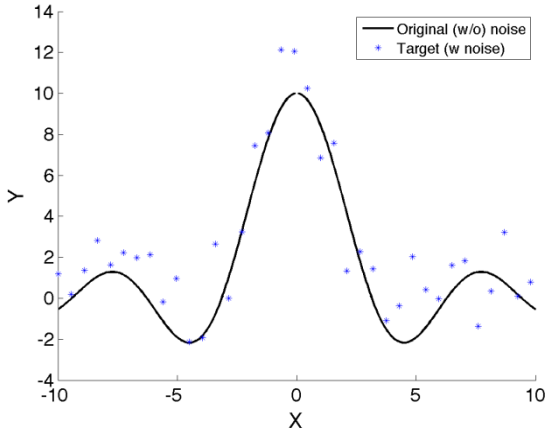


Figure 4.4. Visualization of dataset2

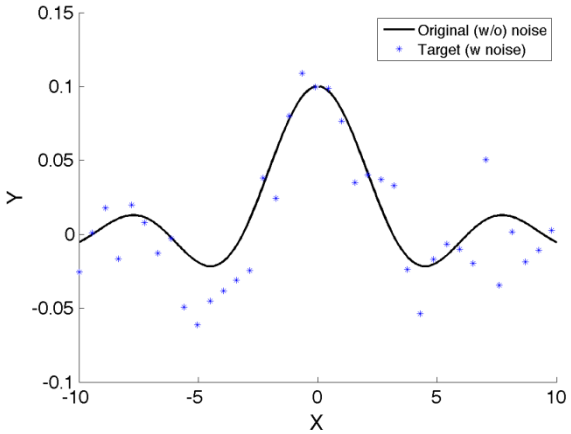


Figure 4.5. Visualization of dataset3

We describe the experimental procedure on three datasets. As mentioned in the previous section, the kernel function is RBF and hyper-parameters with their decision boundaries are shown in Table 4.1. The entire dataset consisting of (x_i, y_i), $i = 1, \dots, 37$ is randomly split into five-folds for cross validation. The data samples in the first fold are left for testing, while the rest of the data samples are used for training a prediction model. Once a prediction model is constructed, the test dataset is presented to the model to calculate the prediction values. This process is repeated for the other four-folds and the mean square error for the predicted values. In case of grid search, every set of candidate hyper-parameters is used to construct five-fold cross validated single SVR or bagging SVR. As for PSO, each particle represents a candidate set of hyper-parameters and the objective function is associated with either the five-cross validated single SVR or bagging SVR. The maximum iteration for GPSO and Meta^2 are set to 100 and 200 respectively. Therefore, for a single run of GPSO, the number of SVRs required to be constructed is the product of the number of particles, number of folds, and maximum iteration, which is equivalent to 30×5×100=15,000.

Table 4.3. Final results obtained for single SVR on the three datasets

	MSE		
Dataset	Grid search	GPSO	Meta^2
1	0.0428	0.0391	0.0391
2	2.6854	2.2465	2.2466
3	4.4047e-04	5.4901e-04	5.7443e-04

We briefly report the results of grid search, GPSO, and Meta^2 for single SVR in order to compare the performance improvement of bagging SVR. Table 4.3. reports the best prediction accuracy obtained by each of the approaches. Note that, for GPSO and

Meta^2, the best results from the ten repetitions are reported. Regarding bagging SVR, GPSO seems to be able to select hyper-parameters that provide better prediction accuracy than grid search for dataset1 and dataset2. Meta^2's results indicate their MSEs are comparable to that of GPSO for the three datasets.

Tables 4.4. through 4.6. report the results obtained by grid search, GPSO, and Meta^2 for bagging SVR on dataset1, dataset2, and dataset3, respectively. Overall, the MSE obtained by GPSO and Meta^2 seem to outperform grid search, while Meta^2 provides comparable results to GPSO and better results in some cases (i.e., ten SVRs for dataset2). Comparing the results of bagging SVR to single SVR, all best bagging SVR models obtained by grid search, GPSO, and Meta^2 outperform the single SVR for dataset1. When grid search is used for 50 SVRs in a bagging model, the MSE is higher than that of single SVR. For dataset2, the same (i.e., bagging SVR provides an improvement in prediction accuracy) holds except when GPSO is used for 30, 40, 45, and 50 SVRs for bagging. Also Meta^2 for 25 SVRs through 50 doesn't seem to provide a better result than single SVR. As for dataset3, GPSO and Meta^2 do not perform better than grid search for single SVR. Also, for bagging SVR, the results obtained by GPSO and Meta^2 do not outperform in many cases. For instance, Meta^2 only outperforms grid search when the number of SVRs is 10 and 45. Similarly GPSO outperforms grid search only when the number of SVRs is 50. For GPSO or Meta^2 to obtain better solutions than grid search for dataset3, one may need to consider using different PSO parameters. For instance, using a larger number of maximum iteration can lead them to obtain better solutions. Table 4.7

summarizes the best results obtained by grid search, GPSO, and Meta^2 for both single SVR and bagging SVR on the three datasets.

Table 4.4. Best results obtained for bagging SVR on dataset1

Number of SVRs	MSE		
	Grid search	GPSO	Meta^2
5	0.0419	0.0364	0.0367
10	0.0416	0.0368	0.0361
15	0.0408	0.0369	0.0365
20	0.0405	0.0322	0.0367
25	0.0423	0.0369	0.0359
30	0.0399	0.0357	0.0366
35	0.0424	0.0369	0.0366
40	0.0412	0.0377	0.0382
45	0.0423	0.0377	0.0367
50	0.0442	0.0339	0.0371

Table 4.5. Best results obtained for bagging SVR on dataset2

Number of SVRs	MSE		
	Grid search	GPSO	Meta2
5	2.5463	2.1255	2.2465
10	2.5131	2.1348	1.9885
15	2.7190	2.2278	2.0934
20	2.6148	2.2126	2.1262
25	2.6814	2.3361	2.2653
30	2.6597	2.2813	2.3530
35	2.7923	2.2395	2.3256
40	2.7122	2.3836	2.3115
45	2.7646	2.3033	2.3139
50	2.7893	2.3590	2.2465

Table 4.6. Best results obtained for bagging SVR on dataset3

Number of SVRs	MSE		
	Grid search	GPSO	Meta^2
5	4.3496e-04	4.9450e-04	4.8398e-04
10	4.4613e-04	4.7434e-04	4.3422e-04
15	4.4736e-04	4.7887e-04	4.9765e-04
20	4.3308e-04	4.4941e-04	4.8732e-04
25	4.4186e-04	4.5668e-04	4.6098e-04

30	4.4316e-04	4.7753e-04	4.9705e-04
35	4.3741e-04	4.5778e-04	4.3924e-04
40	4.2295e-04	4.7684e-04	4.4853e-04
45	4.3906e-04	4.4308e-04	4.3372e-04
50	4.4372e-04	4.3526e-04	4.6839e-04

Table 4.7. Best results obtained by grid search, GPSO, and Meta^2 for the three datasets

		Dataset1		Dataset2		Dataset3	
		Number of SVRs	MSE	Number of SVRs	MSE	Number of SVRs	MSE
Grid search	Single SVR	1	0.0428	1	2.6854	1	4.4047e-04
	Bagging SVR	30	0.0399	10	2.5131	40	4.2295e-04
GPSO	Single SVR	1	0.0391	1	2.2465	1	5.4901e-04
	Bagging SVR	20	0.0322	5	2.1255	50	4.3526e-04
Meta^2	Single SVR	1	0.0391	1	2.2466	1	5.7443e-04
	Bagging SVR	25	0.0359	10	1.9885	45	4.3372e-04

Remark that Meta^2 aims to construct a bagging SVR model by finding an optimal set of hyper-parameters using PSO with meta-modeling. The meta-modeling approach approximates the objective function of the optimization problem to reduce the computational cost in constructing bagging SVR models for candidate solutions. Due to this reason, we aim to obtain a comparable result to GPSO with a reduced computational cost. Figures 4.6 through 4.8 show the mean number of bagging SVR model constructed and the mean elapsed time from the ten repetitions for both GPSO and Meta^2. As mentioned earlier, 3,000 fitness function evaluations are performed for GPSO, which is indicated as a red line in the figures on the left. These 3,000 fitness function evaluations require to construct (3,000 × K-folds × number of SVRs for bagging) SVR models. For example, the number of SVRs constructed for five SVRs in GPSO is equal to 75,000. On

the other hand, Meta^2 performed about 2,200 through 2,400, 2,400 through 2,800, and 2,000 through 2,300 fitness function evaluations for dataset1, dataset2, and dataset3, respectively. For example, the mean number of SVRs constructed for five SVRs in Meta^2 for dataset1 is about 55,000, which requires 20,000 less SVRs in the PSO run.

On the right side of Figures 4.6 through 4.8, the mean elapsed time in minute is shown. Solid bars and lines represent the results of GPSO and empty bars represent Meta^2. Roughly speaking, the mean time increases for both GPSO and Meta^2 with all three datasets as the number of SVRs increases.

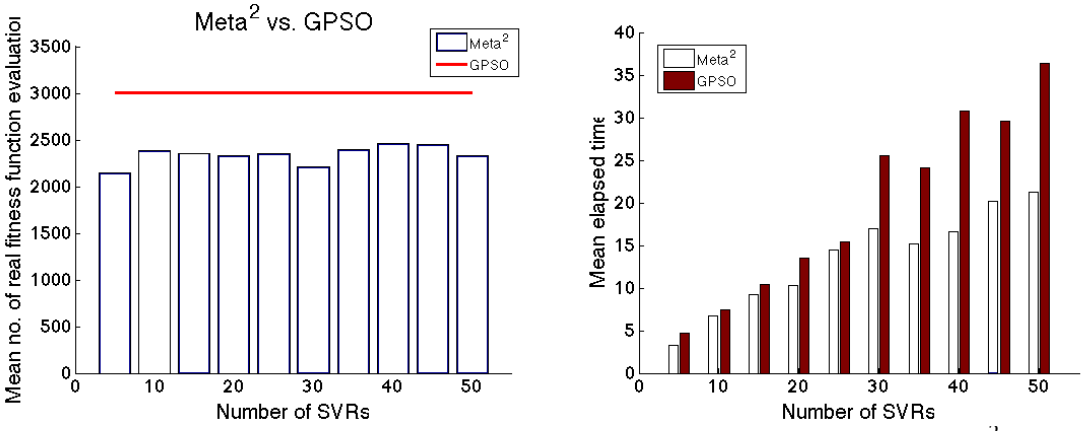


Figure 4.6. Mean number of fitness function evaluations for GPSO and Meta^2 (left) and mean elapsed time in minute (right) for dataset1

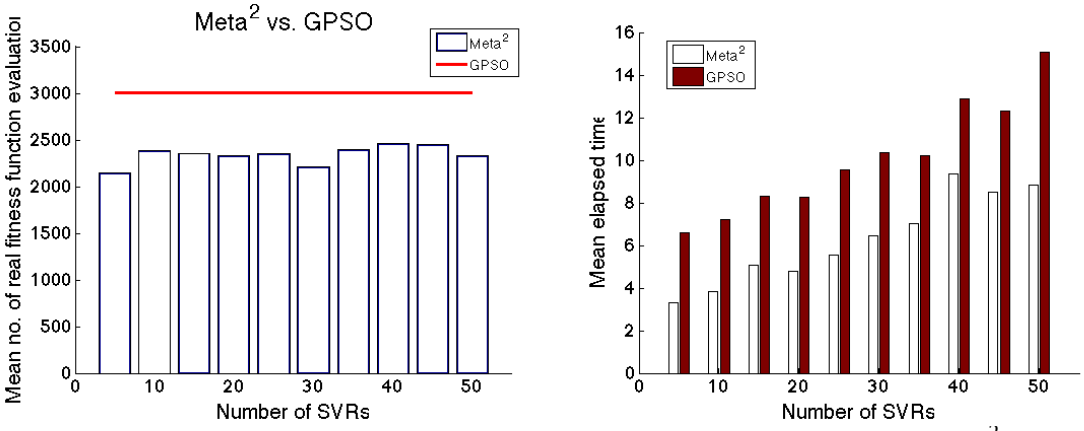


Figure 4.7. Mean number of fitness function evaluations for GPSO and Meta^2 (left) and mean elapsed time in minute (right) for dataset2

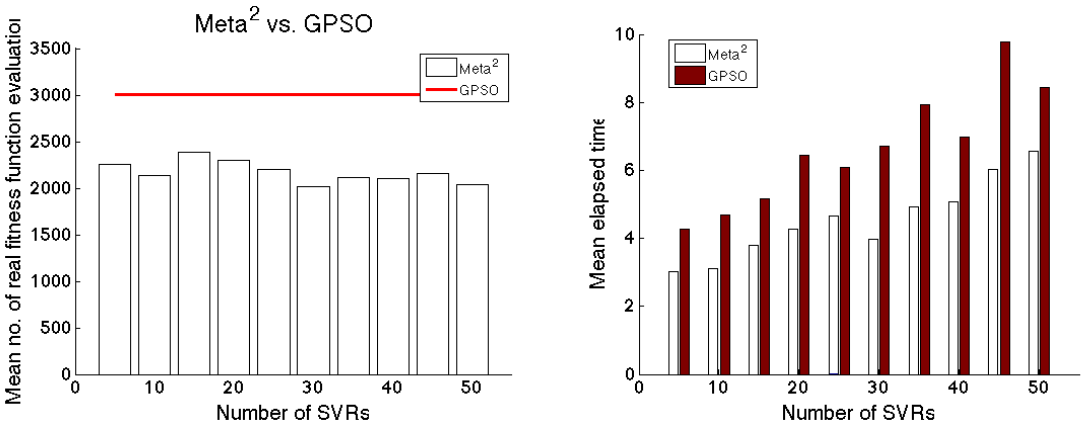


Figure 4.8. Mean number of fitness function evaluations for GPSO and Meta^2 (left) and mean elapsed time in minute (right) for dataset3

In order to confirm that Meta^2 provides comparable results with a reduction in the computation time to GPSO, which does not use meta-modeling, we conduct *t*-tests between the results obtained by the ten repetitions for GPSO and Meta^2. The *t*-tests are performed on the prediction accuracy and computation time. Note, for each number of SVRs, GPSO and Meta^2 construct bagging SVR models ten times and their prediction performance and computation times are used for the *t*-tests. The significance level for the t-tests is set to 5%.

The alternative hypothesis for prediction accuracy, MSE, as H_a^{MSE}, which is "given a number of SVRs for bagging the prediction accuracy obtained by GPSO and Meta^2 are different". As for the computation time, the alternative hypothesis H_a^{time} is "the time taken to complete Meta^2 is less than GPSO." Both types of *t*-tests assume that the results from GPSO and Meta^2 have unequal variances. The significance level used for the tests is 5%. Table 4.8. shows the p-values for these *t*-tests. Therefore, in an ideal case for Meta^2, the *t*-tests should support to not reject the null hypothesis for prediction accuracy and to reject the null hypothesis for the computation time. That way, we will be able to

conclude Meta^2 can provide comparable solutions compared to GPSO while reducing the computation time.

Table 4.8. p-values for *t*-tests between GPSO and Meta^2 in their solution quality and computation time

Number of SVRs	Dataset1		Dataset2		Dataset3	
	H_a^{MSE}	H_a^{time}	H_a^{MSE}	H_a^{time}	H_a^{MSE}	H_a^{time}
5	0.9449	**0.0101**	0.9892	**0.0000**	0.2686	**0.0000**
10	0.6588	0.2620	0.2521	**0.0000**	0.9965	**0.0000**
15	0.6762	0.2633	0.1922	**0.0000**	0.2210	**0.0000**
20	0.3863	0.1027	0.3902	**0.0000**	0.0588	**0.0000**
25	0.2026	0.3843	0.5652	**0.0031**	0.7800	**0.0022**
30	0.9149	**0.0236**	0.4232	**0.0029**	0.3805	**0.0000**
35	0.8632	**0.0275**	0.5668	**0.0000**	0.1381	**0.0000**
40	0.3105	**0.0024**	0.9948	**0.0289**	0.9607	**0.0040**
45	0.2543	**0.0338**	0.6392	**0.0012**	0.7596	**0.0067**
50	0.9762	**0.0193**	0.1187	**0.0049**	0.9970	**0.0248**

**bold letters indicate that the p-values are small enough to reject the null hypothesis*

In conclusion, we notice all the p-values for the *t*-tests on MSE do not reject the null hypothesis. Therefore, it is likely that the quality of solution obtained by Meta^2 is comparable to GPSO for all the cases (i.e., different numbers of SVRs for the three datasets). On the other hand, the p-values for elapsed time reject the null hypothesis. In particular, Meta^2 is likely to provide less computation time for dataset1 when the number of SVRs is 5, 30, 35, 40, 45, and 50. As for dataset2 and dataset3, Meta^2 will likely provide less computation time for all cases. Note that the number of bagging SVR model constructed in Meta^2 is less than GPSO as mentioned earlier. We claim that Meta^2 is capable of providing comparable solutions with a reduced computational cost.

4.3.2. RSW quality dataset

The RSW quality dataset consists of 1,280 data samples, each of which represents different welding parameters. Overall, there are two sets of weldment design-related features corresponding to two different materials and one set of welding process related features that describe the welding process performed on these two materials. The dataset consists of 16 input features. The welding quality is described by nugget width in this experiment. Particularly in this data set, three types of different materials are considered. The data are obtained by physical testing conducted by welding experts. Table 4.9. shows the welding design and process variables.

Table 4.9. Features for the welding quality dataset

Design features	
Material	Coating HDG
Thickness	Coating weight
Coating EG	Surface class
Process features	
Weld force	Weld current
Min button DIA of stack-up	Weld time
Response output	
Nugget width	

We construct bagging SVR models to predict the welding quality (i.e., nugget width) using the dataset. There is a large amount of noise interrupting the task of constructing a reliable prediction model. We briefly describe how the noise exists in the dataset. In this problem, we define noise as a data sample that represents the same welding parameters with different nugget width in millimeter.

Figure 4.9. shows how noise is distributed in the dataset. The way that the plot is drawn is as follows. Firstly, we group the data samples based on the 16 welding

parameter features, so that in a group all the welding parameters are the same. This results in 262 different groups of welding parameters, which corresponds to the x-axis in Figure 4.9. The groups are sorted in an increasing order of their nugget width. The number of data samples in each group ranges from two to as many as 232. Secondly, the mean and standard deviation of nugget width are calculated for each group.

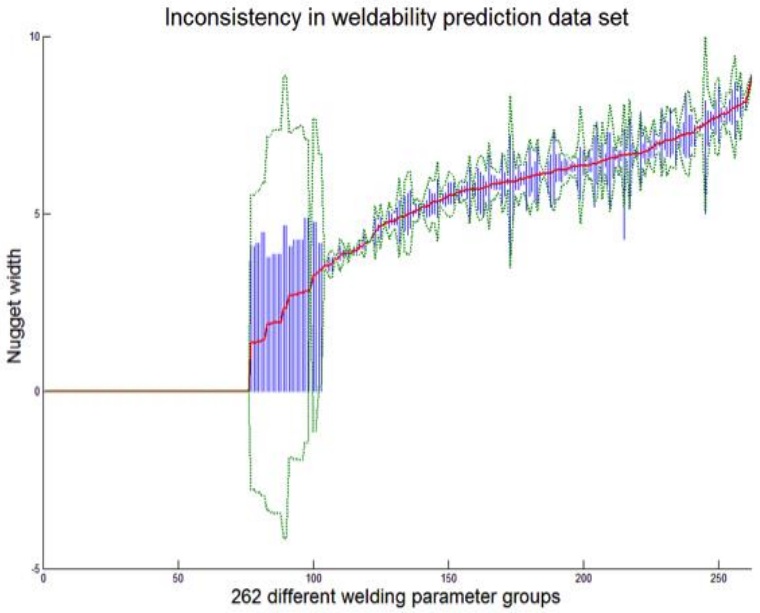


Figure 4.9. Illustration of noise in the RSW quality dataset

Then, the groups are ordered in ascending order of the mean nugget width. Blue, red, and green lines represent the variability within a group, mean nugget width, and approximate confidence interval. The confidence interval is calculated by assuming the nugget width within each group is normally distributed with a 95% significance level. Roughly speaking, the welding parameter groups from zero to 76 have no variability; in other words, there is no noise. These welding parameters may simply lead to a bad weld judging from the data. Groups from 77 to 103 are where the most significant noise exists in this dataset. The rest of groups from 104 to the end do not seem to have significant

noise. Most of the variations seem to be the random variations caused by the welding process.

The experiment procedure remains the same as for the three artificial datasets in the previous section. However, for this dataset, three-fold cross validation is used. The maximum iteration is set to 30 and 100 for GPSO and Meta^2 respectively. Table 4.10. reports the prediction accuracy obtained by grid search, GPSO, and Meta^2 for single SVR respectively. Note that, for GPSO and Meta^2, the best results from the ten repetitions are reported. The results show a slight improvement with both GPSO and Meta^2 compared to grid search.

Table 4.10. Final results obtained for single SVR on the welding quality dataset

	Grid search	GPSO	Meta^2
MSE	1.7704	1.7629	1.7637

Table 4.11. Best results obtained for bagging SVR on the welding quality dataset

Number of SVRs	MSE		
	Grid search	GPSO	Meta^2
5	1.7364	1.7185	1.7218
10	1.7269	1.7009	1.7111
15	1.7317	1.7217	1.7169
20	1.7431	1.7116	1.7185
25	1.7291	1.7201	1.7091
30	1.7264	1.7190	1.7209
35	1.7290	1.7307	1.7174
40	1.7302	1.7259	1.7259
45	1.7285	1.7280	1.7209
50	1.7328	1.7191	1.7199

Similarly, Table 4.11. reports that the results for bagging SVR. GPSO and Meta^2 seem to outperform grid search similar to single SVR. As confirmed earlier for the three datasets, Meta^2 seems to provide comparable results to GPSO with all different numbers

of SVRs. Better results are obtained for some cases (i.e., 15, 25, 35, and 45 SVRs). We also notice that bagging SVR improves the prediction accuracy of a single SVR for this dataset.

Table 4.12. summarizes the best results obtained by grid search, GPSO, and Meta^2 for both single SVR and bagging SVR on the RSW quality dataset. As mentioned in the previous section for the three artificial datasets, due to the approximation procedure in meta-modeling, we expect comparable solutions to GPSO with a reduced computational cost by using Meta^2. Figure 4.10. shows the mean number of bagging SVR models constructed and the mean elapsed time from the ten repetitions for both GPSO and Meta^2. For this dataset, the number of fitness function evaluations is limited to 900 for GPSO, which is indicated as a red line in the figure on the left. The 900 fitness function evaluations require to construct (900 × K-folds × number of SVRs) SVR models. For instance, when the number of SVRs in bagging is five, GPSO constructs 13,500 whereas Meta^2 constructs about 12,000 SVRs. Also, the figures on the right show the mean elapsed time in minute. Solid bars and lines represent the results of GPSO and empty bars represent Meta^2.

Table 4.12. Best results obtained by grid search, GPSO, and Meta^2 for the welding quality dataset

		Number of SVRs	MSE
Grid search	Single SVR	1	1.7704
	Bagging SVR	30	1.7264
GPSO	Single SVR	1	1.7629
	Bagging SVR	10	1.7009
Meta^2	Single SVR	1	1.7637
	Bagging SVR	25	1.7091

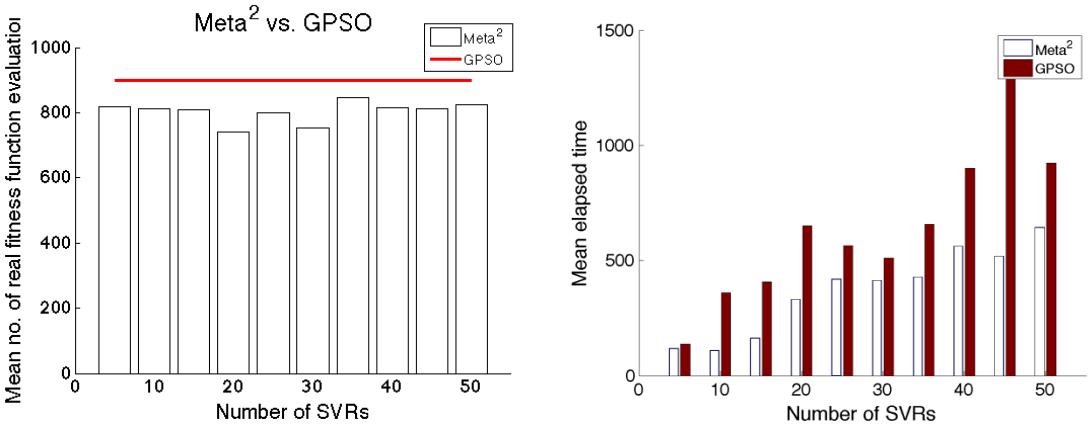


Figure 4.10. Mean number of fitness function evaluations for GPSO and Meta^2 (left) and mean elapsed time in minute (right) for the welding quality dataset

To confirm that Meta^2 provides comparable results to GPSO, *t*-tests are conducted using the final solutions obtained by GPSO and Meta^2 from the ten repetitions. Again, the alternative hypothesis for prediction accuracy, H_a^{MSE}, is given a number of SVRs for bagging the prediction accuracy obtained by GPSO and Meta^2 are different. The alternative hypothesis for the computation time, H_a^{time}, is that the time taken for Meta^2 is less than GPSO. Note, for each number of SVRs, GPSO and Meta^2 construct bagging SVR models ten times and their prediction performance and computation times are used for the *t*-tests. We assume that the solutions obtained by GPSO and Meta^2 have unequal variances. The significance level is 5%. The results of these *t*-tests are shown in Table 4.13. As mentioned earlier for the three artificial datasets, it is ideal if the *t*-tests can support to not reject the null hypothesis for prediction accuracy and to reject the null hypothesis for the computation time.

We conclude that all the p-values for the *t*-tests on MSE do not reject the null hypothesis. Therefore, it is likely that Meta^2 can obtain comparable solutions to GPSO all

the cases (i.e., different numbers of SVRs). As for the computation time, the null hypothesis is rejected based on the *t*-test results for most of the cases, except that there seems to be no significant computation time reduction when the number of SVRs is equal to 5, 40, and 50. The *t*-test results indicate that Meta^2 is likely to provide a reduction on the computation time. Therefore, these results show that Meta^2 is capable of providing comparable solutions with a reduced computational cost for this welding quality dataset.

Table 4.13. p-values for *t*-tests between GPSO and Meta^2 in their solution quality and computation time for the welding quality dataset

Number of SVRs	H_a^{MSE}	H_a^{time}
5	0.2501	0.1660
10	0.6164	**0.0000**
15	0.8524	**0.0000**
20	0.1780	**0.0000**
25	0.9149	**0.0121**
30	0.6595	**0.0293**
35	0.0613	**0.0402**
40	0.5497	0.0559
45	0.7492	**0.0011**
50	0.7201	0.1425

**bold letters indicate the p-values are small enough to reject the null hypothesis*

4.4. Integration of Meta^2 with a design optimization and decision making system

We illustrate how Meta^2 can be utilized to assist design activities using modeFRONTIER which is a design optimization and decision making tool. In general, multidisciplinary approaches and multi-objective decisions are involved in the design process. These multi-disciplinary approaches include computer-aided design and manufacturing (CAD/CAM), statistical analysis such as design of experiment (DOE) techniques, and visualization whereas multi-objective decisions can be accomplished by optimization and predictive modeling techniques. modeFRONTIER allows designers to

integrate these various activities into workflows where each activity can be examined investigated with the graphical user interface (GUI). That way, designers and engineers investigate the design solutions and the effects of conflicting objectives in order to identify the design process.

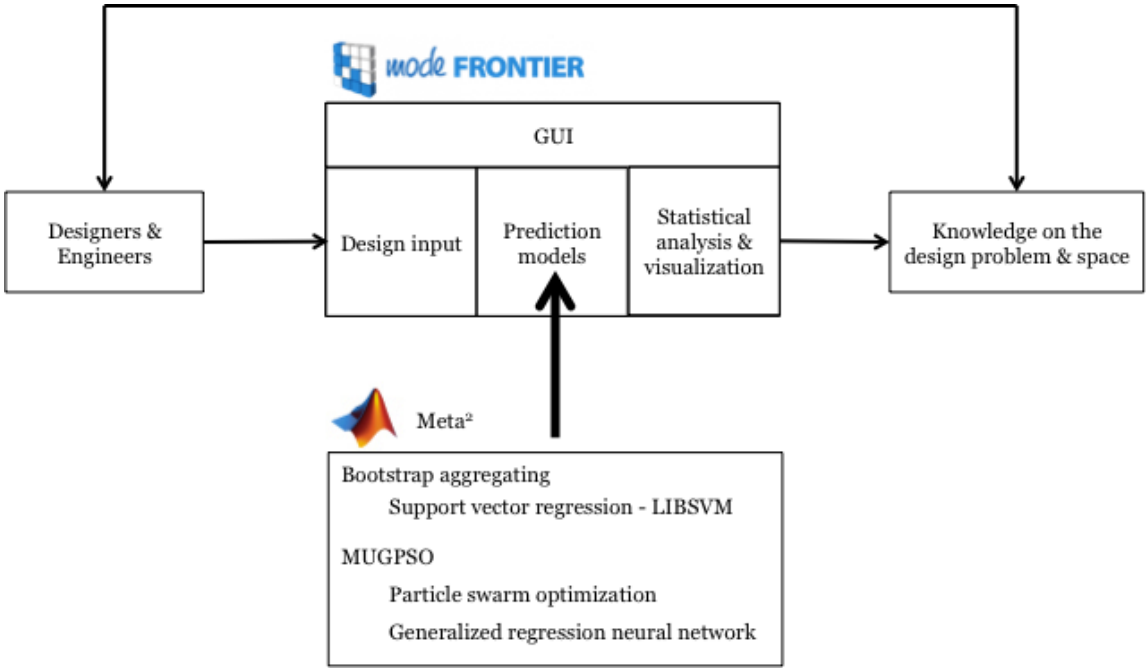


Figure 4.11. Illustration of the integration process flow

Fig 4.11. describes a process flow of our developed Meta2 framework integrated with the modeFRONTIER tool. Designers and engineers can use the tool to understand and work on their design space. modeFRONTIER provides the GUI to define various design alternatives as input in the entire design workflow. The prediction models constructed by Meta2 are obtained by running MATLAB scripts. These MATLAB scripts are connected to the modeFRONTIER workflow so that the prediction models can be used to analyze the design alternatives designers and/or engineers are interested in identifying further. One can summarize the results of the analysis and visualize the results

using a number of statistical analysis and visualization techniques included in modeFRONTIER.

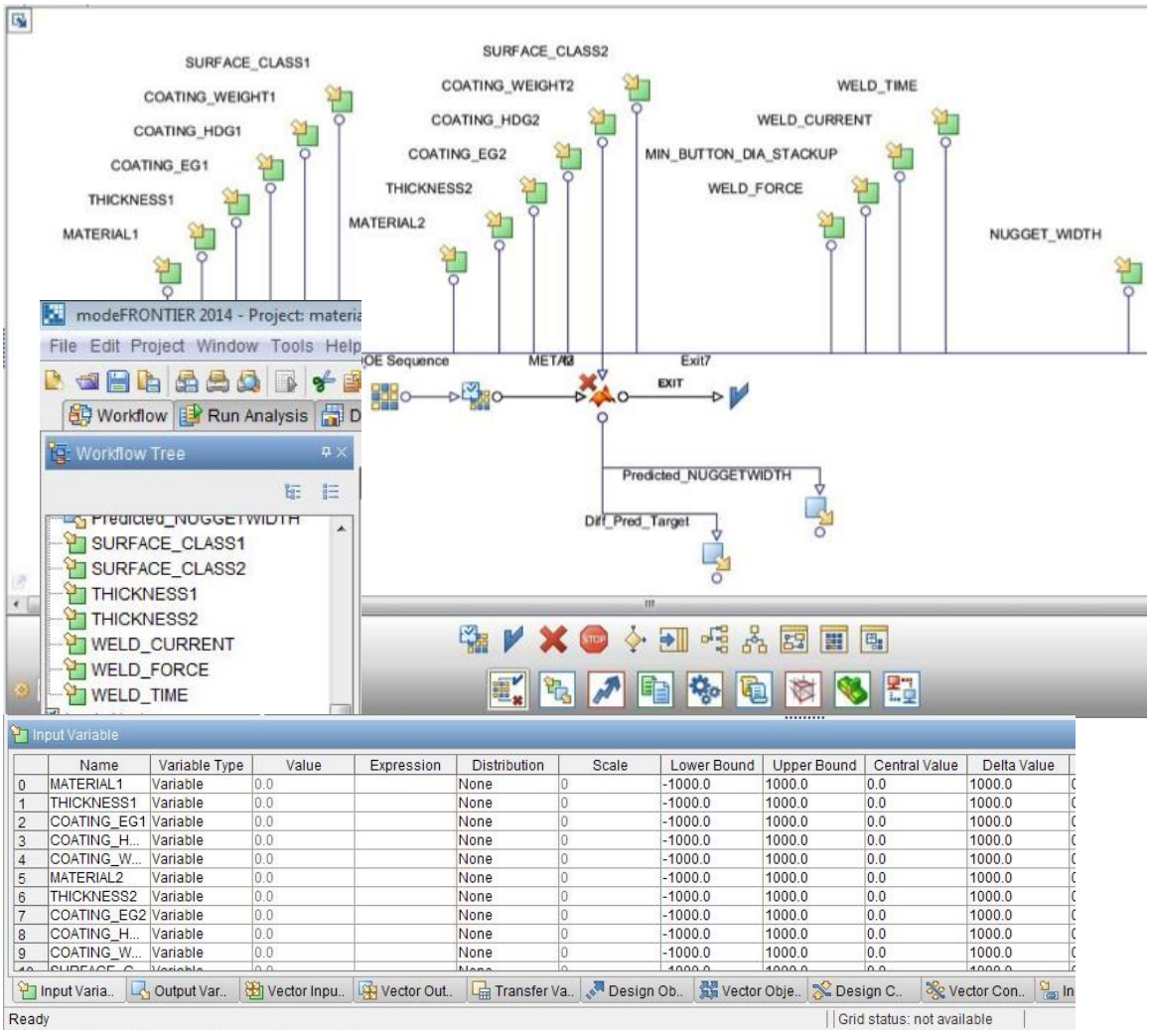


Figure 4.12. An example modeFRONTIER workflow for material selection using prediction models constructed by Meta[2]

Figures 4.12. through 4.15. show screenshots taken from the modeFRONTIER tool. Figure 4.12. illustrates a workflow created in modeFRONTIER for the above process. The input features listed in Table 4.9. are defined as input variable nodes.

The DOE Sequence node can read the dataset from a file such as text or Excel. Also, as the name indicates, a number of DOE techniques are available to generate input data. Then, the workflow moves forward to the MATLAB script node where the MATLAB scripts for our Meta^2 implementation are linked. Finally, the prediction values for input data are defined as an output variable named Predicted_NUGGETWIDTH. In addition, to illustrate the data analysis features included in modeFRONTIER, another output variable named Diff_Pred_Target is included in the workflow, which calculates the difference between the target and predicted values of an input data.

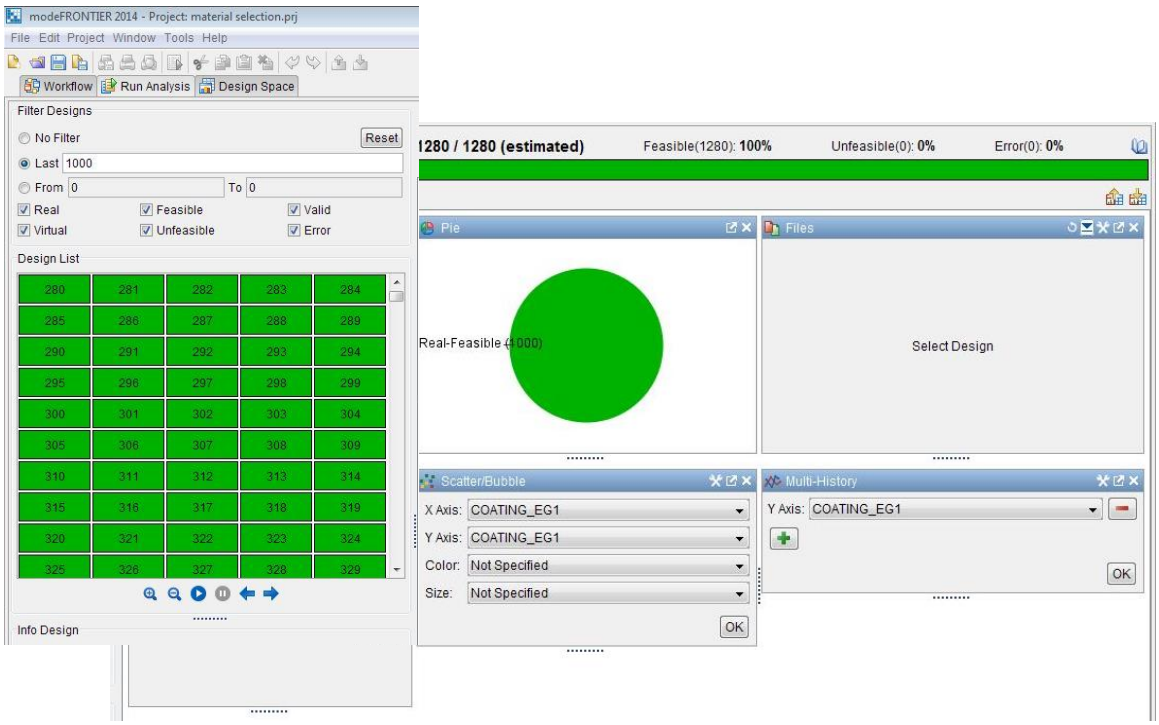


Figure 4.13. Run analysis feature in modeFRONTIER

Once a workflow is properly set up, one can run the workflow in the Run Analysis tab as shown in Figure 4.13. Essentially, using this feature, the workflow can repeat and also users can selectively run the workflow on certain input data (i.e., design

alternatives). After the run, the results can be viewed in the Design Space tab as shown in Figure 4.14. All the input and output data are listed in tables. For example, the Designs Table includes the input data and as well as the output. In addition, in the Design Space tab, users can apply statistical analysis, visualization, and other features included in modeFRONTIER.

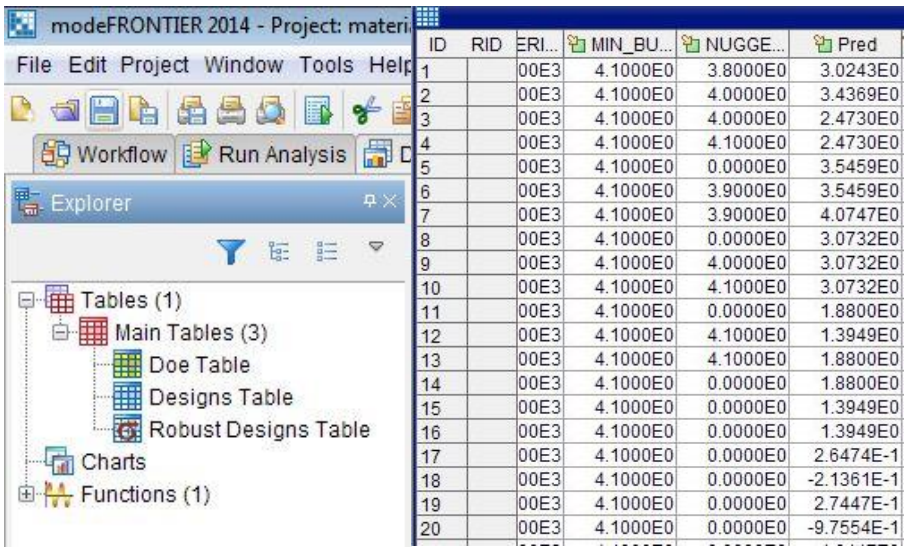


Figure 4.14. Decision space explorer in modeFRONTIER

As an illustration, we apply clustering analysis on Diff_Pred_Target as mentioned earlier. The reason we take the difference between the target and predicted values is because as shown in Figure 4.9. noisy data are likely to have prediction values that are far away from the target value. Therefore, by looking at the difference, one can easily identify and examine noisy data in the dataset. Figure 4.15. shows the results of hierarchical clustering applied to the difference values. We cluster them into three clusters. Cluster0 is where the difference between the target and predicted values are the largest, which means these are most likely noisy data. Note that the within cluster

distance is the smallest for Cluster0 and as well as the number of data. This can be an indication that the variety of noisy data is closely distributed each other.

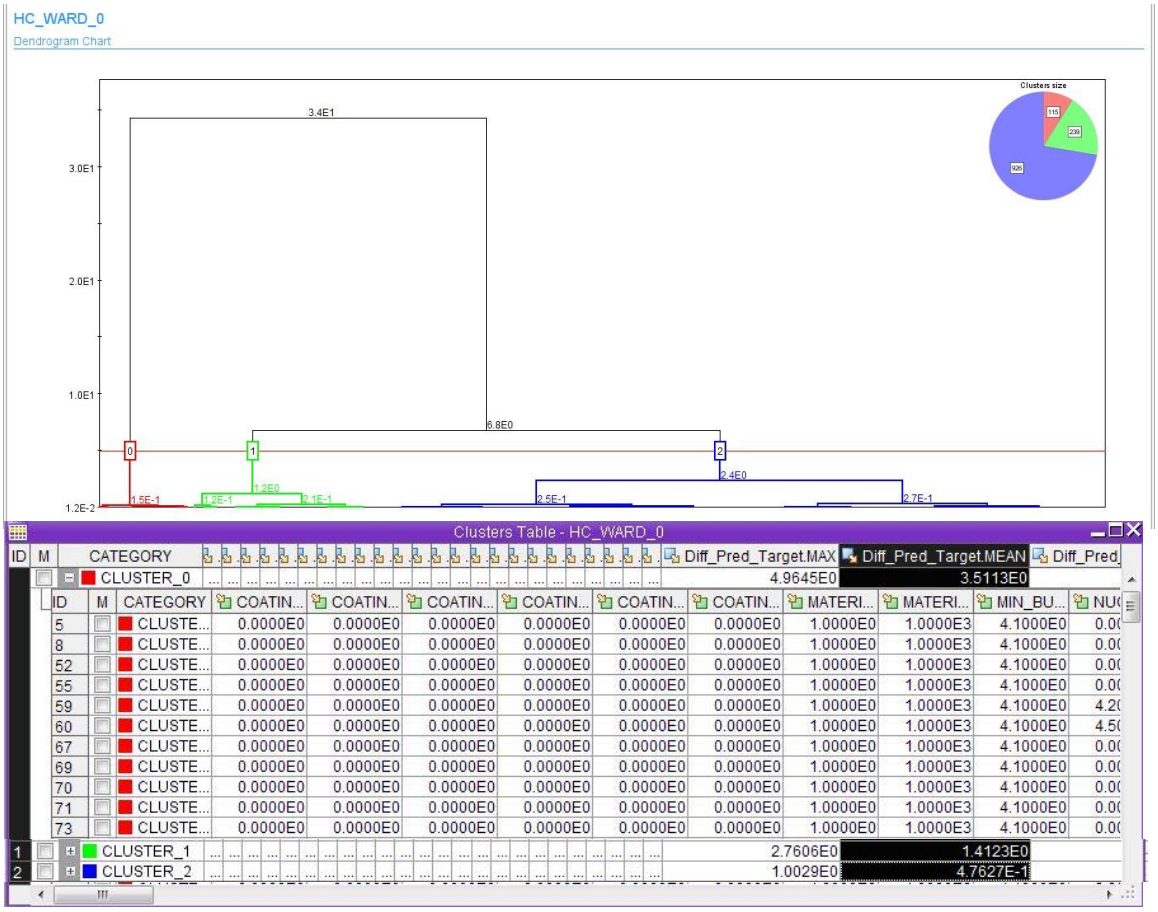


Figure 4.15. Clustering analysis in modeFRONTIER

4.5. Summary

In this chapter, we consider constructing bagging prediction models for noisy manufacturing data with SVR as the base learning algorithm. The problem is examined using our proposed prediction framework called Meta^2. The hyper-parameter optimization is solved using PSO with meta-modeling in Meta^2 in order to obtain quality solutions with a reduction in the overall computational cost. The proposed approach is

performed on three different datasets artificially generated with noise and as well as on a noisy RSW quality dataset. The results obtained by Meta^2 reveal that the solution quality is comparable to GPSO in all the cases. In most of the cases, Meta^2 is likely to provide a significant computation time reduction. Also, the experimental results show that SVR is an appropriate choice as the base learning algorithm for bagging as it provides an improvement on the prediction accuracy. In addition, modeFRONTIER is used to integrate the prediction models constructed by Meta^2. An illustration of how these prediction models can be integrated to support design decision makings using the features included in modeFRONTIER such as statistical analysis.

CHAPTER 5 CONCLUSION AND FUTURE RESEARCH DIRECTION

In this research, we consider the noisy data problems often found in manufacturing process data. Instead of removing noisy data, we aim to identify an approach to construct more precise prediction models without removing them. A novel prediction modeling framework, called Meta^2, is proposed and examined a number of noisy datasets. The main contribution of using this framework is that one can construct prediction models for noisy data with improved prediction accuracy and less computational cost.

The Meta^2 prediction modeling framework can be used for noisy data where the noise cannot be removed before constructing prediction models. The prediction models are constructed using bagging SVR. We have identified related research to bagging models and SVRs. In regards to bagging, related issues discussed include selecting the base learning algorithm for bagging and number of learning algorithm. As for SVRs,

issues related to hyper-parameters selection are discussed, which includes the effect of hyper-parameters and kernel functions.

The hyper-parameters for the SVRs in bagging models are selected using PSO and MUGPSO. The experiments conducted on datasets in this research reveal that using SVR to construct bagging models can improve the prediction accuracy on noisy data. Also, MUGPSO provides comparable quality solutions with reduced computational cost. As an illustration, we describe a scenario of how prediction models constructed by Meta2 can be integrated with design activities using modeFRONTIER.

The future research work is manifold. Regarding bagging, other ensembles such as boosting, over-bagging, and under-bagging can be examined for other types of data challenges. For example, imbalance classification problems are often seen. We assume the prediction performance can be improved using one of the ensemble methods. Meta^2 can be further expanded for such an approach. In this research, we also only consider SVR and RBF kernels. For classification with different kinds of data, SVM and other kernels may be necessary to include in the optimization layer. This will leave a more difficult optimization problem since the nature of decision variables include both integer and real values. Based on related research in the literature, we assumed that using a set of hyper-parameters for all the SVRs in bagging is appropriate. However, the choice is not limited to such an approach. Different or dynamic sets of hyper-parameters can be applied. Identifying different approaches to this extent and their results is a direction to continue this research.

In regards to MUGPSO, we will work on improving the quality of final solutions. This can be done by many different approaches. One of them is to add local meta-models. The effect of smoothing factor for GRNN can be also examined further. The current version of MUGPSO simply stores data samples by cutting off using a threshold. More sophisticated approaches in determining what data samples should be stored is also another area that can be studied in future research. Finally, the meta-modeling methods are compared to ten benchmark functions with only a 30-dimension. We leave a study on the performance on different dimensionalities as a future work. With respect to Meta^2, it will be a valuable research direction to identify how many real fitness function evaluations (i.e., bagging model construction) can be replaced by meta-models in relation to maintaining comparable quality solutions.

In this research, we consider the prediction performance in terms of the mean square error. We report that using bagging SVR on noisy data improves the prediction performance of a prediction model. However, in general, small changes in such noisy datasets have high effect on the prediction performance while affecting the data characteristic in the dataset. Therefore, considering the sensitivity or robustness of a prediction model to such changes in noisy datasets will be a valuable research area.

Finally, an important area of future research is how to utilize the prediction models constructed by Meta^2. As an illustration, we show modeFRONTIER integrates the MATAB scripts for Meta^2. This should be further researched to elaborate so that design processes and activities can be aided. In addition, identifying the causes of such noisy data and approaches will be an interesting research topic.

APPENDIX A. LIST OF GENERAL NOTATIONS

n: The number of data samples in a dataset.

n_k: The number of data samples in the kth fold dataset.

d: The dimensionality of input vector **x.**

X: A set of input data samples. For example, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ where each vector $\mathbf{x}_i$ represents the *i*th data sample and $\mathbf{x}_i \in \mathbb{R}^d$.

$\mathbf{x}_i$: The *i*th data sample

x_{ik}: The kth element in the sample $\mathbf{x}_i$

y_i: *i*th class label or response value respectively for a classification and regression problem.

T: A dataset which contains $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$.

$f(\mathbf{x})$: The original function between **x** and y.

$\hat{f}(\mathbf{x})$: A prediction function to approximate $f(\mathbf{x})$.

APPENDIX B. LIST OF SUPPORT VECTOR MACHINE AND SUPPORT VECTOR REGRESSION RELATED NOTATIONS

w: The coefficient vector that determines a hyperplane.

b: The threshold that determines a hyperplane.

ρ: The margin between two classes.

ξ: The slack variable for non-separable problems.

C: The penalty coefficient.

ε: ε precision for ε-SVR.

$K(\,)$: A kernel function

APPENDIX C. LIST OF BOOTSTRAP AGGREGATING RELATED NOTATIONS

n': The number of data samples in a bootstrapped dataset.

T^l: The *l*th bootstrapped dataset.

L: The number of bootstrapped datasets.

$\hat{f}^l(\mathbf{x})$: The prediction function constructed from the dataset T^l.

$\hat{f}_{bag}(\mathbf{x})$: A prediction function constructed by bagging.

$\varphi(\,)$: An aggregating function

APPENDIX D. LIST OF KRIGING RELATED NOTATIONS

$g(\mathbf{x})$: A global model of the original function

$Z(\mathbf{x})$: A local deviation from the global model $g(\mathbf{x})$

β: The underlying coefficients of the polynomial

$\hat{\beta}$: The estimated parameter for β

σ^2: The process variance

$R(\mathbf{x}_i, \mathbf{x}_j)$: The correlation between any two data samples $\mathbf{x}_i$ and $\mathbf{x}_j$

$r^T(\mathbf{x}) = [R(\mathbf{x}, \mathbf{x}_1), \dots, R(\mathbf{x}, \mathbf{x}_n)]^T$

θ_k: The Gaussian correlation function parameter

APPENDIX E. LIST OF POLYNOMIAL REGRESSION RELATED NOTATIONS

β: The coefficient terms

n_{coef}: The number of coefficient terms

APPENDIX F. LIST OF RADIAL BASIS FUNCTION NETWORK RELATED NOTATIONS

γ_i: The radial basis function for the *i*th hidden neuron

w_o: The bias term

w_i: The weight coefficient associated with γ_i

c_j: The *j*th center

APPENDIX G. LIST OF GENERALIZED REGRESSION NEURAL NETWORK RELATED NOTATIONS

σ: The smoothing factor

$D_i^2(\mathbf{x}, \mathbf{x}_i)$: The distance between $\mathbf{x}$ and $\mathbf{x}_i$

APPENDIX H. LIST OF PARTICLE SWARM OPTIMIZATION RELATED NOTATIONS

N_s: The number of particles in a swarm

D: The dimension of problem

$\mathbf{P}_l$: The *l*th particle

$\mathbf{L}_l^t$: The local best solution for the *l*th particle at iteration *t*

$\mathbf{G}^t$: The global best solution at iteration *t*

$\mathbf{V}_l^t$: The velocity vector for the *l*th particle at iteration *t*

w: Inertia weight

c_1 and c_2: Acceleration coefficients for the local and global best solutions respectively

r_1 and r_2: Uniform random numbers from [0,1] for the local and global best solutions respectively

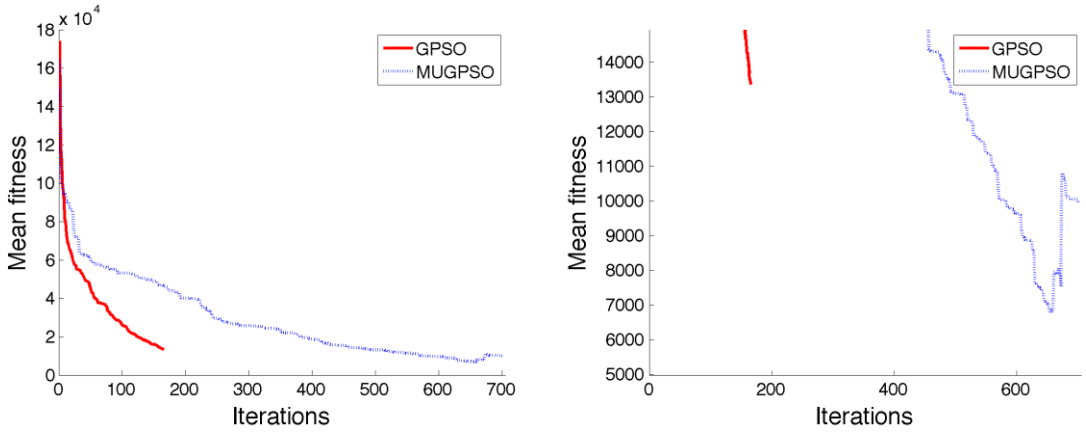
w_{start}: The starting value of the inertia weight

w_{end}: The end values of the inertia weight

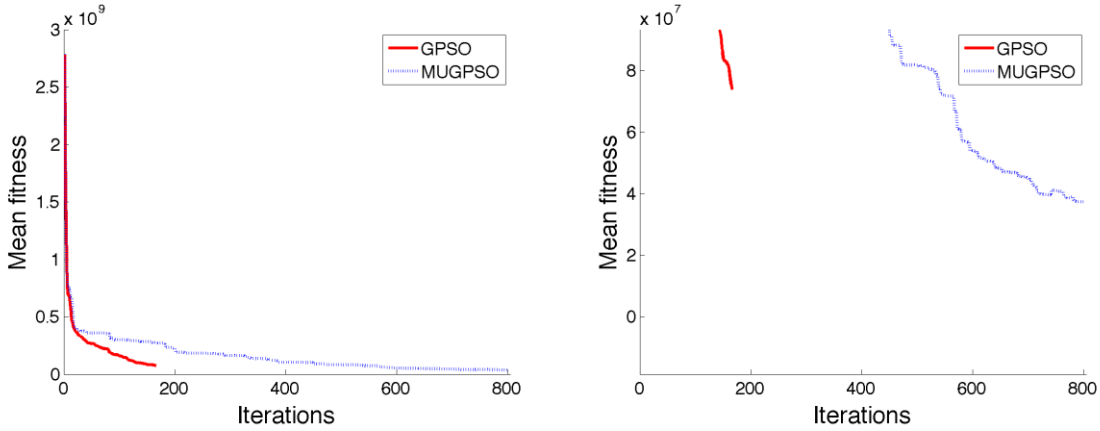
APPENDIX I. STATISTICAL RESULTS OF SOLUTIONS OBTAINED BY GPSO AND MUGPSO WITH A LIMIT OF 20,000 REAL FITNESS FUNCTION EVALUATIONS

	Algorithms	Global optimum	Best	Worst	Mean	SD	Number of Evaluations
F1	GPSO	-4.50e+02	-4.47e+02	-4.29e+02	-4.42e+02	5.30e+00	19980
	MUGPSO		-4.49e+02	-4.28e+02	-4.49e+02	0.62e+00	19397
F2	GPSO	-4.50e+02	3.78e+03	1.10e+04	7.25e+03	2.12e+03	19980
	MUGPSO		2.33e+02	3.19e+03	1.47e+03	9.05e+02	19587
F3	GPSO	-4.50e+02	1.24e+07	5.04e+07	3.26e+07	1.15e+07	19980
	MUGPSO		1.05e+07	2.61e+07	1.69e+07	5.18e+06	18828
F4	GPSO	-4.50e+02	6.55e+03	2.11e+04	1.56e+04	4.66e+03	19980
	MUGPSO		2.87e+03	1.75e+04	8.97e+03	4.95e+03	18727
F5	GPSO	-3.10e+02	4.59e+03	6.33e+03	5.54e+03	5.78e+02	19980
	MUGPSO		3.03e+03	6.38e+03	4.32e+03	9.60e+02	18206
F6	GPSO	3.90e+02	5.08e+03	8.57e+04	3.10e+04	2.48e+04	19980
	MUGPSO		5.58e+02	1.48e+04	2.65e+03	4.46e+03	19738
F7	GPSO	-1.80e+02	-1.78e+02	-1.75e+02	-1.77e+02	0.89e+00	19980
	MUGPSO		-1.78e+02	-1.74e+02	-1.77e+02	1.25e+00	18920
F8	GPSO	-1.40e+02	-1.18e+02	-1.18e+02	-1.18e+02	0.04e+00	19980
	MUGPSO		-1.19e+02	-1.18e+02	-1.18e+02	0.04e+00	18585
F9	GPSO	-3.30e+02	-2.97e+02	-2.58e+02	-2.81e+02	1.21e+01	19980
	MUGPSO		-3.01e+02	-2.73e+02	-2.84e+02	9.32e+00	19252
F10	GPSO	-3.30e+02	-1.37e+02	-8.18e+01	-1.07e+02	1.59e+01	9960
	MUGPSO		-2.11e+02	-8.40e+01	-1.37e+02	4.54e+01	9692

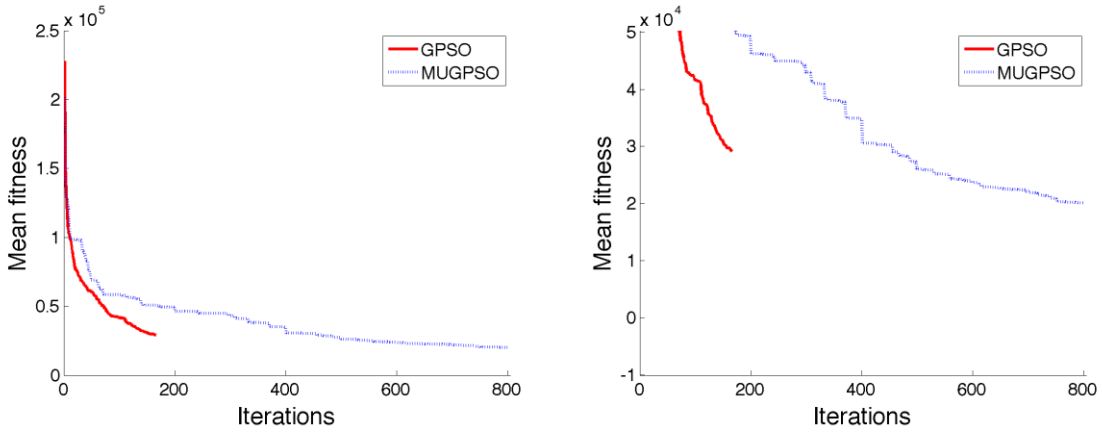
APPENDIX J. CONVERGENCE PROFILE OF GPSO AND MUGPSO ON F2, F3, F4, F8, F9, and F10



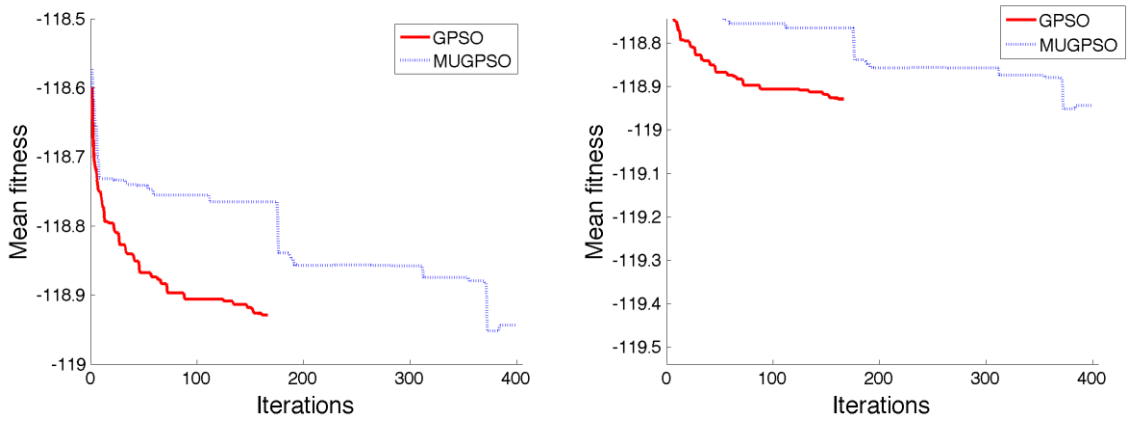
APPENDIX J-a. Convergence profile of GPSO and MUGPSO on F2. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).



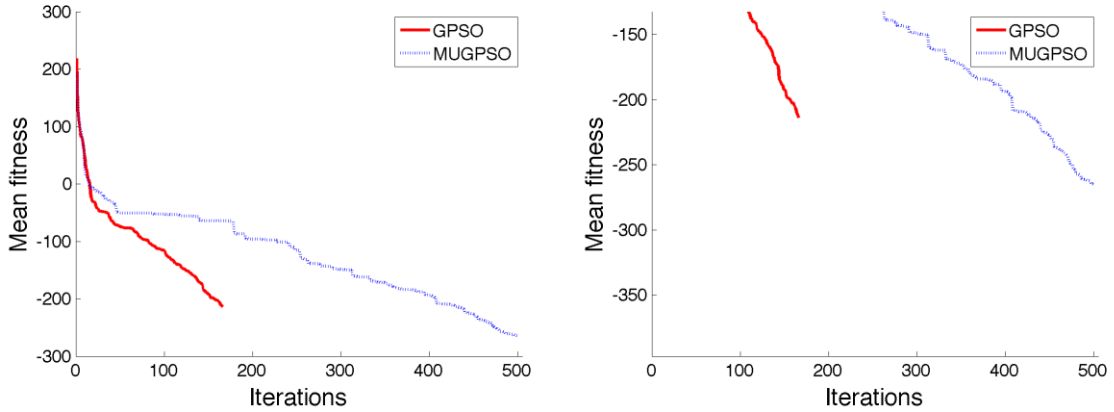
APPENDIX J-b. Convergence profile of GPSO and MUGPSO on F3. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).



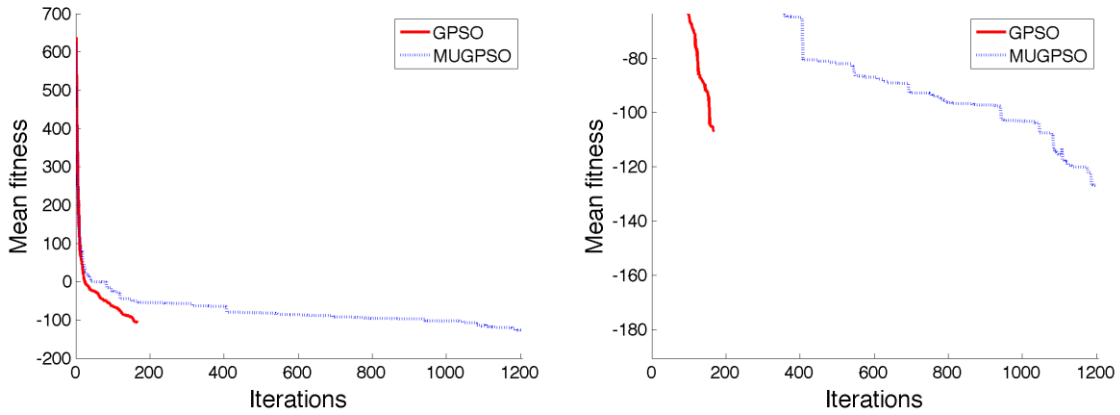
APPENDIX J-c. Convergence profile of GPSO and MUGPSO on F4. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).



APPENDIX J-d. Convergence profile of GPSO and MUGPSO on F8. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).



APPENDIX J-e. Convergence profile of GPSO and MUGPSO on F9. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).



APPENDIX J-f. Convergence profile of GPSO and MUGPSO on F10. The mean solutions visited at every iteration (left) and mean solutions visited zoomed in near the end of iterations (right).

APPENDIX K. MATLAB SCRIPTS FOR META^2 IMPLEMENTATION

```
clear;
load('welddata.mat');

%Decision boundaries for C, epsilon, gamma (rbf kernel)
lb = [2^-3 2^-8 2^-15];
ub = [2^15 2^1 2^3];

%Specify swarm size
D = numel(lb);
Nswarm = numel(lb) * 10;

%Initialize variables to store results
rep = 10;
Allx=cell(1,numel(5:5:50));
Allfval=zeros(rep,numel(5:5:50));
Allelaptime=zeros(rep,numel(5:5:50));
Allfitcount=cell(rep,numel(5:5:50));
Allgbestvals=cell(rep,numel(5:5:50));

%Repeatedly run Meta2 different number of SVR
j=1;
for N_SVR=5:5:50,
```

```
tempAllx=zeros(rep,D);

   for i=1:rep,
      tic
      [gbest,gbestval,allgbestval,fitcount] = MUGPSO(D, Nswarm, max_iter, lb, ub,
input, target, artfdata.CVO, N_SVR);
      Allelaptime(i,j) = toc;
      Allgbestvals{i,j} = allgbestval;
      Allfitcount{i,j} = fitcount;
      tempAllx(i,:)= gbest;
      Allfval(i,j)=gbestval;
      fprintf('completed - dataset: %s, iteration: %d, rep: %d, N_SVR: %d, Time taken:
%.2fmin.\n',datasetname, max_iter, i, N_SVR, Allelaptime(i,j)/60);

   end
   Allx{j} = tempAllx;
   j=j+1;
end

%Save results to file
save(strcat(datasetname,'_T',iter,'_MUGPSO_baggingSVR_result.mat'),'Allx','Allfval','Al
lelaptime','Allfitcount','Allgbestvals');


function [gbest,gbestval,gbest_values,allgbestval, fitcount]= MUGPSO(Dimension,
Particle_Number, MUGPSOparam, VRmin, VRmax, input, target, CVO, N_SVR)

%PSO parameter initialization
ps = Particle_Number;
D = Dimension;
%Acceleration constants c1 and c2
c = [MUGPSOparam.C1 MUGPSOparam.C2];
%inertia weight 0.9 to 0.5
iwt = MUGPSOparam.wstart-
(1:MUGPSOparam.maxiter).*(0.5./MUGPSOparam.maxiter);
%initialize velocities
if length(VRmin)==1
   VRmin=repmat(VRmin,1,D);
```

```
    VRmax=repmat(VRmax,1,D);
end
mv = 0.5*(VRmax-VRmin);
VRmin = repmat(VRmin,ps,1);
VRmax = repmat(VRmax,ps,1);
Vmin = repmat(-mv,ps,1);
Vmax = -Vmin;
vel = Vmin+2.*Vmax.*rand(ps,D);

%Initia swarm using Latin hypercube design
pos = lhsdesign(ps, D);
pos = icdf('unif', pos, VRmin, VRmax);

%Evaulate fitness on the initial swarm
e = zeros(ps,1);
e_values = cell(ps,1);

evalfuncname=str2func('fitness_baggingSVR');
for p = 1:ps,
   [e(p,1) e_values{p}] = feval(evalfuncname, input, target, CVO, pos(p,:), N_SVR);
end

%Store some particles
metamodel_input = pos;
metamodel_target = e;
metamodel_inputdb=[];
metamodel_targetdb=[];
spread=MUGPSOparam.smoothingfactor;

%Count number of fitness evaluation on particles
fitcount = ps;

%Update local best and their fitness values
pbest = pos;
pbestval = e;
pbestval_values = e_values;
%Update global best and the fitness value
[gbestval, gbestind] = min(pbestval);
```

```
gbest_values = pbestval_values{gbestind};
gbest = pbest(gbestind,:);
gbestrep = repmat(gbest,ps,1);
allgbestval=gbestval;

%Iterate PSO process
i=2;
for i=2:MUGPSOparam.maxiter

   %Update velocities
   tempvel = c(1).*rand(ps,D).*(pbest-pos)+c(2).*rand(ps,D).*(gbestrep-pos);

   vel = iwt(i).*vel + tempvel;
   %limit velocities to the range
   %velocities higher or lower than the range are replaced by the min max
   %of the range
   vel = (vel>Vmax).*Vmax + (vel<=Vmax).*vel;
   vel = (vel<Vmin).*Vmin + (vel>=Vmin).*vel;

   %update swarm
   pos = pos+vel;
   %particles higher or lower than the range are replaced by the min max
   %of the range +- 0.25 to avoid particles placed on the boundary
   pos = ((pos>=VRmin)&(pos<=VRmax)).*pos...
      +(pos<VRmin).*(VRmin+0.25.*(VRmax-VRmin).*rand(ps,D))...
      +(pos>VRmax).*(VRmax-0.25.*(VRmax-VRmin).*rand(ps,D));

   %use the nearest data samples from the db to construct the meta-model
   if numel(metamodel_inputdb) >0
      closestones = knnsearch( metamodel_inputdb, pos,'K',1);
      globalinput = [metamodel_inputdb(closestones(:,1),:)];
      globaltarget = [metamodel_targetdb(closestones(:,1),:)];
   else
      closestones=[];
      globalinput = metamodel_input;
      globaltarget = metamodel_target;

   end
```

```
   %construct a meta-model and evaluate current particles
   metamodel = newgrnn(globalinput', globaltarget', spread );
   e = sim(metamodel, pos')';

   %update the local best solution and fitness
   tmp = (pbestval < e);
   tempind = find(tmp==0);

   if numel(tempind) >= 1
      temppos = pos(tempind, :);
      tempe = zeros(size(temppos,1),1);
      tempe_values = cell(size(temppos,1),1);
      if N_SVR==1
         for p = 1:size(temppos,1),
            [tempe(p,:) tempe_values{p}]= feval(evalfuncname, input, target, CVO,
temppos(p,:));
         end
      else
         for p = 1:size(temppos,1),
             [tempe(p,:) tempe_values{p}] = feval(evalfuncname, input, target, CVO,
temppos(p,:), N_SVR);
         end
      end
      globaldb = abs( (e(tempind) - tempe)./tempe );
      globaldbind = find(globaldb>0.01);
      metamodel_inputdb = [metamodel_inputdb; temppos(globaldbind,:)];
      metamodel_targetdb = [metamodel_targetdb; tempe(globaldbind)];

      e(tempind) = tempe;
      for t=1:numel(tempind);
         e_values{tempind(t)} = tempe_values{t};
      end
      fitcount = [fitcount, fitcount(end)+ numel(tempind)];
   else
      tempe = zeros(ps,1);
      tempe_values = cell(ps,1);
      if N_SVR==1
```

```
            for p = 1:ps,
                [tempe(p,:) tempe_values{p}]= feval(evalfuncname, input, target, CVO,
pos(p,:));
            end
        else
            for p = 1:ps,
                [tempe(p,:) tempe_values{p}] = feval(evalfuncname, input, target, CVO,
pos(p,:), N_SVR);
            end
        end
        globaldb = abs( (e - tempe)./tempe );
        globaldbind = find(globaldb>0.01);

        metamodel_inputdb = [metamodel_inputdb; pos(globaldbind,:)];
        metamodel_targetdb = [metamodel_targetdb; tempe(globaldbind)];
        numel(find(globaldbind ==1));

        e=tempe;
        e_values = tempe_values;
        fitcount = [fitcount, fitcount(end)+ numel(e)];
    end

    tmp = (pbestval <e);
    temp = repmat(tmp, 1, D);
    pbest = temp.*pbest+(1-temp).*pos;
    pbestval = tmp.*pbestval+(1-tmp).*e;
    ind = find(tmp==0);
    for t=1:numel(ind);
        pbestval_values{ind(t)} = e_values{ind(t)};
    end

    %Update the global best solution and fitness
    [gbestval,tmp]=min(pbestval);
    gbest_values = pbestval_values{tmp};
    gbest=pbest(tmp,:);
    gbestrep=repmat(gbest,ps,1);
    allgbestval(i) = gbestval;
```

```
end

end


function [ fitness,cvOutput ] = fitness_baggingSVR( input, target, CVO, hyperparams,
N_SVR )

%Candidate solution
C = hyperparams(1);
eps = hyperparams(2);
gamma = hyperparams(3);

%K fold cross validation
K= CVO.NumTestSets;

%Output variable to store target and prediction values
testingoutputs=cell(K,1);
testingtargets=cell(K,1);
%Construct bagging models using K fold cross validation
parfor k=1:K,
   trIdx = CVO.training(k);
   teIdx = CVO.test(k);

   trinputs = input(trIdx,:);
   trtargets = target(trIdx,:);
   teinputs = input(teIdx,:);
   tetargets = target(teIdx,:);

   trainInd = find(trIdx==1);
   testInd = find(teIdx==1);
   opts = sprintf('%s %s %s %s %s %s %s','-q -s 3 -t 2','-p',num2str(eps),'-g',
num2str(gamma),'-c',num2str(C));

   [testingoutputs{k}]=baggingSVR(trinputs,trtargets,teinputs,tetargets, N_SVR, opts);
   testingtargets{k} = tetargets;
end
%Output variable: target and prediction values
```

```
cvOutput = [cell2mat(testingoutputs), cell2mat(testingtargets)];

%MSE
fitness = mse(cvOutput(:,1)-cvOutput(:,2));
end

function [ bagging_testoutput ] = baggingSVR( traindata, traintargetdata, testdata,
testtargetdata, N_SVR, SVRopts )
%Generate bootstrapped datasets
[bootstat, bootsam] = bootstrp(N_SVR,@mean, traindata);

%Train SVR on each bootstrapped dataset
parfor (i=1:N_SVR)
   model(i) = svmtrain(traintargetdata(bootsam(:,i),:), traindata(bootsam(:,i),:), SVRopts);
   testoutput(:,i) = svmpredict(testtargetdata, testdata, model(i),'-q');
end

%Aggregate SVRs
bagging_testoutput = mean(testoutput,2);

end
```

REFERENCES

Andrés, E., Salcedo-Sanz, S., Monge, F., and Pérez-Bellido, A. (2012). "Efficient aerodynamic design through evolutionary programming and support vector regression algorithms." *Expert Systems with applications*, Elsevier, 39(12), 10700–10708.

Arciszewski, T., and De Jong, K. (2001). "Evolutionary computation in civil engineering: research frontiers." Civil and structural engineering computing: 2001, Saxe-Coburg Publications, 161–184.

Ayat, N.-E., Cheriet, M., and Suen, C. Y. (2005). "Automatic model selection for the optimization of SVM kernels." *Pattern Recognition*, Elsevier, 38(10), 1733–1745.

Bauer, E., and Kohavi, R. (1999). "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants." *Machine learning*, Springer, 36(1-2), 105–139.

Bengio, Y. (2000). "Gradient-based optimization of hyperparameters." *Neural Computation*, MIT Press, 12(8), 1889–1900.

Białobrzewski, I. (2008). "Neural modeling of relative air humidity." *Computers and electronics in agriculture*, Elsevier, 60(1), 1–7.

Bird, S., and Li, X. (2010). "Improving local convergence in particle swarms by fitness approximation using regression." Computational Intelligence in Expensive Optimization Problems, Springer, 265–293.

Breiman, L. (1996). "Bagging predictors." *Machine learning*, Springer, 24(2), 123–140.

Burges, C. J. (1998). “A tutorial on support vector machines for pattern recognition.” *Data mining and knowledge discovery*, Springer, 2(2), 121–167.

Chang, C.-C., and Lin, C.-J. (2011). “LIBSVM: a library for support vector machines.” *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM, 2(3), 27.

Chang, M.-W., and Lin, C.-J. (2005). “Leave-one-out bounds for support vector regression model selection.” *Neural Computation*, MIT Press, 17(5), 1188–1222.

Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. (2002). “Choosing multiple parameters for support vector machines.” *Machine learning*, Springer, 46(1-3), 131–159.

Chatterjee, A., Pulasinghe, K., Watanabe, K., and Izumi, K. (2005). “A particle-swarm-optimized fuzzy-neural network for voice-controlled robot systems.” *Industrial Electronics, IEEE Transactions on*, IEEE, 52(6), 1478–1489.

Chen, K.-Y. (2007). “Forecasting systems reliability based on support vector regression with genetic algorithms.” *Reliability Engineering & System Safety*, Elsevier, 92(4), 423–432.

Chen, S., Wang, W., and Van Zuylen, H. (2009). “Construct support vector machine ensemble to detect traffic incident.” *Expert Systems with applications*, Elsevier, 36(8), 10976–10986.

Clarke, S. M., Griebsch, J. H., and Simpson, T. W. (2005). “Analysis of support vector regression for approximation of complex engineering analyses.” *Journal of Mechanical Design*, American Society of Mechanical Engineers, 127(6), 1077–1087.

Clerc, M., and Kennedy, J. (2002). “The particle swarm-explosion, stability, and convergence in a multidimensional complex space.” *Evolutionary Computation, IEEE Transactions on*, IEEE, 6(1), 58–73.

Currit, N. (2002). “Inductive regression: overcoming OLS limitations with the general regression neural network.” *Computers, environment and urban systems*, Elsevier, 26(4), 335–353.

Dias, J., Rocha, H., Ferreira, B., and do Carmo Lopes, M. (2013). “A genetic algorithm with neural network fitness function evaluation for IMRT beam angle optimization.” *Central European Journal of Operations Research*, Springer, 1–25.

Dietterich, T. G. (2000). “An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization.” *Machine learning*, Springer, 40(2), 139–157.

Enders, C. K. (2010). “Applied missing data analysis.” Guilford Press.

Escalante, H. J., Montes, M., and Sucar, L. E. (2009). “Particle swarm model selection.” *The Journal of Machine Learning Research*, JMLR. org, 10, 405–440.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). “Working set selection using second order information for training support vector machines.” *The Journal of Machine Learning Research*, JMLR. org, 6, 1889–1918.

Fang-shu, C., and Jian-Chao, Z. (2009). “An effective intelligent algorithm for stochastic optimization problem.” Control and Decision Conference, 2009. CCDC'09. Chinese, IEEE, 3197–3202.

Fleming, P. J., and Purshouse, R. C. (2002). “Evolutionary algorithms in control systems engineering: a survey.” *Control engineering practice*, Elsevier, 10(11), 1223–1241.

Gamberger, D., Lavrač, N., and Džeroski, S. (1996). “Noise elimination in inductive concept learning: A case study in medical diagnosis.” Algorithmic Learning Theory, Springer, 199–212.

Gheyas, I. A., and Smith, L. S. (2010). “Feature subset selection in large dimensionality domains.” *Pattern Recognition*, Elsevier, 43(1), 5–13.

Gomide, F. (n.d.). “Fuzzy Clustering in Fitness Estimation Models for Genetic Algorithms and Applications.” Fuzzy Systems, 2006 IEEE International Conference on, IEEE, 1388–1395.

Graham, J. W. (2009). “Missing data analysis: Making it work in the real world.” *Annual review of psychology*, Annual Reviews, 60, 549–576.

Gräning, L., Jin, Y., and Sendhoff, B. (2007). “Individual-based management of meta-models for evolutionary optimization with application to three-dimensional blade optimization.” Evolutionary Computation in Dynamic and Uncertain Environments, Springer, 225–250.

Guo, X., Yang, J., Wu, C., Wang, C., and Liang, Y. (2008). “A novel LS-SVMs hyper-parameter selection based on particle swarm optimization.” *Neurocomputing*, Elsevier, 71(16), 3211–3215.

Hendtlass, T. (2007). “Fitness estimation and the particle swarm optimisation algorithm.” Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, IEEE, 4266–4272.

Hibbert, D. B. (2012). “Experimental design in chromatography: a tutorial review.” *Journal of chromatography B*, Elsevier, 910, 2–13.

Huang, C.-M., Lee, Y.-J., Lin, D. K., and Huang, S.-Y. (2007). “Model selection for support vector machines via uniform design.” *Computational Statistics & Data Analysis*, Elsevier, 52(1), 335–346.

Ivanciuc, O. (2007). “Applications of support vector machines in chemistry.” *Reviews in computational chemistry*, Wiley; 1999, 23, 291.

Jeng, J.-T. (2005). “Hybrid approach of selecting hyperparameters of support vector machine for regression.” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, IEEE, 36(3), 699–709.

Jin, R., Chen, W., and Simpson, T. W. (2001). “Comparative studies of metamodelling techniques under multiple modelling criteria.” *Structural and Multidisciplinary Optimization*, Springer, 23(1), 1–13.

Jin, Y. (2005). “A comprehensive survey of fitness approximation in evolutionary computation.” *Soft computing*, Springer, 9(1), 3–12.

Jin, Y. (2011). “Surrogate-assisted evolutionary computation: Recent advances and future challenges.” *Swarm and Evolutionary Computation*, Elsevier, 1(2), 61–70.

Jin, Y., and Sendhoff, B. (2004). “Reducing fitness evaluations using clustering techniques and neural network ensembles.” Genetic and Evolutionary Computation–GECCO 2004, Springer, 688–699.

Kapp, M. N., Sabourin, R., and Maupin, P. (2009). "A PSO-based framework for dynamic SVM model selection." Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 1227–1234.

Kavaklioglu, K. (2011). "Modeling and prediction of Turkey's electricity consumption using Support Vector Regression." *Applied Energy*, Elsevier, 88(1), 368–375.

Khoshgoftaar, T. M., Van Hulse, J., and Napolitano, A. (2011). "Comparing boosting and bagging techniques with noisy and imbalanced data." *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, IEEE, 41(3), 552–568.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., and Bang, S.-Y. (2002). "Support vector machine ensemble with bagging." Pattern recognition with support vector machines, Springer, 397–408.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., and Yang Bang, S. (2003). "Constructing support vector machine ensemble." *Pattern Recognition*, Elsevier, 36(12), 2757–2767.

Kim, H.-S., and Cho, S.-B. (2001). "An efficient genetic algorithm with less fitness evaluation by clustering." Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, IEEE, 2, 887–894.

Kim, I.-S., Jeong, Y., Lee, C., and Yarlagadda, P. (2003). "Prediction of welding parameters for pipeline welding using an intelligent system." *The International Journal of Advanced Manufacturing Technology*, Springer, 22(9-10), 713–719.

Kim, M.-J., and Kang, D.-K. (2012). "Classifiers selection in ensembles using genetic algorithms for bankruptcy prediction." *Expert Systems with applications*, Elsevier, 39(10), 9308–9314.

Kleijnen, J. P. (1986). "Statistical tools for simulation practitioners." Marcel Dekker, Inc.

Kusiak, A., and Salustri, F. (2007). "Computational intelligence in product design engineering: review and trends." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, 37(5), 766–778.

Lim, D., Ong, Y.-S., Jin, Y., and Sendhoff, B. (2007). "A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation." Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, 1288–1295.

Lin, S.-W., Ying, K.-C., Chen, S.-C., and Lee, Z.-J. (2008). "Particle swarm optimization for parameter determination and feature selection of support vector machines." *Expert Systems with applications*, Elsevier, 35(4), 1817–1824.

Lins, I. D., Moura, M. D. C., Zio, E., and Droguett, E. L. (2012). "A particle swarm-optimized support vector machine for reliability prediction." *Quality and Reliability Engineering International*, Wiley Online Library, 28(2), 141–158.

Little, R. J., and Rubin, D. B. (1987). "Statistical analysis with missing data." Wiley New York, 539.

Liu, H. (2010). "Instance selection and construction for data mining." Springer-Verlag.

Liu, H., and Motoda, H. (2002). "On issues of instance selection." *Data mining and knowledge discovery*, Springer, 6(2), 115–130.

Lu, X., Tang, K., and Yao, X. (2011). “Classification-assisted differential evolution for computationally expensive problems.” Evolutionary Computation (CEC), 2011 IEEE Congress on, IEEE, 1986–1993.

Mao, W., Yan, G., Dong, L., and Hu, D. (2011). “Model selection for least squares support vector regressions based on small-world strategy.” *Expert Systems with applications*, Elsevier, 38(4), 3227–3237.

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). “Comparison of three methods for selecting values of input variables in the analysis of output from a computer code.” *Technometrics*, Taylor & Francis, 21(2), 239–245.

Melville, P., Shah, N., Mihalkova, L., and Mooney, R. J. (2004). “Experiments on ensembles with missing and noisy data.” Multiple Classifier Systems, Springer, 293–302.

Mendes, R., Cortez, P., Rocha, M., and Neves, J. (2002). “Particle swarms for feedforward neural network training.” Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on, IEEE, 2, 1895–1899.

Moser, G., and Serpico, S. B. (2009). “Automatic parameter optimization for support vector regression for land and sea surface temperature estimation from remote sensing data.” *Geoscience and Remote Sensing, IEEE Transactions on*, IEEE, 47(3), 909–921.

Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., and Kittler, J. (2010). “A review of instance selection methods.” *Artificial Intelligence Review*, Springer, 34(2), 133–143.

Opitz, D., and Maclin, R. (1999). “Popular Ensemble Methods: An Empirical Study.” *Journal of Artificial Intelligence Research*, 11, 169–198.

Oyang, Y.-J., Hwang, S.-C., Ou, Y.-Y., Chen, C.-Y., and Chen, Z.-W. (2005). “Data classification with radial basis function networks based on a novel kernel density estimation algorithm.” *IEEE transactions on neural networks*, IEEE, 16(1), 225–236.

Pal, M. (2008). “Ensemble of support vector machines for land cover classification.” *International journal of remote sensing*, Taylor & Francis, 29(10), 3043–3049.

Pal, S. K., Bandyopadhyay, S., and Ray, S. S. (2006). “Evolutionary computation in bioinformatics: A review.” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, 36(5), 601–615.

Pal, S., Pal, S. K., and Samantaray, A. K. (2008). “Artificial neural network modeling of weld joint strength prediction of a pulsed metal inert gas welding process using arc signals.” *Journal of materials processing technology*, Elsevier, 202(1), 464–474.

Pan, F., Zhu, P., and Zhang, Y. (2010). “Metamodel-based lightweight design of B-pillar with TWB structure via support vector regression.” *Computers & structures*, Elsevier, 88(1), 36–44.

Park, J.-H., and Seo, K.-K. (2006). “A knowledge-based approximate life cycle assessment system for evaluating environmental impacts of product design alternatives in a collaborative design environment.” *Advanced Engineering Informatics*, Elsevier, 20(2), 147–154.

Park, S.-Y., and Lee, J.-J. (2014). “An efficient differential evolution using speeded-up k-nearest neighbor estimator.” *Soft computing*, Springer, 18(1), 35–49.

Parno, M., Hemker, T., and Fowler, K. (2012). “Applicability of surrogates to improve efficiency of particle swarm optimization for simulation-based problems.” *Engineering optimization*, Taylor & Francis, 44(5), 521–535.

Polikar, R. (2012). “Ensemble learning.” Ensemble Machine Learning, Springer, 1–34.

Praveen, C., and Duvigneau, R. (2009). “Low cost PSO using metamodels and inexact pre-evaluation: Application to aerodynamic shape design.” *Computer Methods in Applied Mechanics and Engineering*, Elsevier, 198(9), 1087–1096.

Regis, R. G. (2014). “Particle swarm with radial basis function surrogates for expensive black-box optimization.” *Journal of Computational Science*, Elsevier, 5(1), 12–23.

Ren, Y., Sun, C., Zeng, J., and Pan, J. (2013). “GRNN-based Prediction Strategy for Particle Swarm Optimization..” *International Journal of Advancements in Computing Technology*, 5(6).

Reyes-Sierra, M., and Coello Coello, C. A. (2005). “A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization.” Evolutionary Computation, 2005. The 2005 IEEE Congress on, IEEE, 1, 65–72.

Robinson, T. J., Borror, C. M., and Myers, R. H. (2004). “Robust parameter design: a review.” *Quality and Reliability Engineering International*, Wiley Online Library, 20(1), 81–101.

Rubin, D. B. (1976). “Inference and missing data.” *Biometrika*, Biometrika Trust, 63(3), 581–592.

Schafer, J. L., and Graham, J. W. (2002). “Missing data: our view of the state of the art..” *Psychological methods*, American Psychological Association, 7(2), 147.

Shi, L., and Rasheed, K. (2010). “A survey of fitness approximation methods applied in evolutionary algorithms.” Computational intelligence in expensive optimization problems, Springer, 3–28.

Shi, Y., and Eberhart, R. (1998a). “A modified particle swarm optimizer.” Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, IEEE, 69–73.

Shi, Y., and Eberhart, R. C. (1998b). “Parameter Selection in Particle Swarm Optimization.” Proceedings of the 7th International Conference on Evolutionary Programming VII, Springer-Verlag, 591–600.

Shieh, M.-D., and Yang, C.-C. (2008). “Classification model for product form design using fuzzy support vector machines.” *Computers & Industrial Engineering*, Elsevier, 55(1), 150–164.

Smola, A. J., and Schölkopf, B. (2004). “A tutorial on support vector regression.” *Statistics and computing*, Springer, 14(3), 199–222.

Sousa, I., and Wallace, D. (2006). “Product classification to support approximate life-cycle assessment of design concepts.” *Technological Forecasting and Social Change*, Elsevier, 73(3), 228–249.

Specht, D. F. (1991). “A general regression neural network.” *IEEE transactions on neural networks*, IEEE, 2(6), 568–576.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). “Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization.” *KanGAL Report*, 2005005.

Sun, C., Jin, Y., Zeng, J., and Yu, Y. (2014). "A two-layer surrogate-assisted particle swarm optimization algorithm." *Soft computing*, Springer, 1–15.

Sun, C., Zeng, J., Pan, J., Xue, S., and Jin, Y. (2013). "A new fitness estimation strategy for particle swarm optimization." *Information sciences*, Elsevier, 221, 355–370.

Tang, Yucheng, and Chen, J. (2009). "Robust design of sheet metal forming process based on adaptive importance sampling." *Structural and Multidisciplinary Optimization*, Springer, 39(5), 531–544.

Tang, Yuanfu, Chen, J., and Wei, J. (2013). "A surrogate-based particle swarm optimization algorithm for solving optimization problems with expensive black box functions." *Engineering optimization*, Taylor & Francis, 45(5), 557–576.

Valentini, G. (2005). "An experimental bias-variance analysis of SVM ensembles based on resampling techniques." *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, IEEE, 35(6), 1252–1271.

Valentini, G., and Dietterich, T. G. (2003). "Low bias bagged support vector machines." ICML, 752–759.

Van den Bergh, F., and Engelbrecht, A. P. (2006). "A study of particle swarm optimization particle trajectories." *Information sciences*, Elsevier, 176(8), 937–971.

Vapnik, V. N. (1995). "Thc Nature of Statistical Learning Theory. Spring-Verlag, New York." Inc.

Vapnik, V. N. (1998). "Statistical learning theory." Wiley.

Venugopal, V., and Narendran, T. (1992). "Neural network model for design retrieval in manufacturing systems." *Computers in Industry*, Elsevier, 20(1), 11–23.

Wang, G. G., and Shan, S. (2007). "Review of metamodeling techniques in support of engineering design optimization." *Journal of Mechanical Design*, American Society of Mechanical Engineers, 129(4), 370–380.

Wang, J., and Wan, W. (2009). "Experimental design methods for fermentative hydrogen production: a review." *International journal of hydrogen energy*, Elsevier, 34(1), 235–244.

Wang, S.-J., Mathew, A., Chen, Y., Xi, L.-F., Ma, L., and Lee, J. (2009). "Empirical analysis of support vector machine ensemble classifiers." *Expert Systems with applications*, Elsevier, 36(3), 6466–6476.

Wu, C.-H., Tzeng, G.-H., and Lin, R.-H. (2009). "A Novel hybrid genetic algorithm for kernel function and parameter optimization in support vector regression." *Expert Systems with applications*, Elsevier, 36(3), 4725–4735.

Yagci, O., Mercan, D., Cigizoglu, H., and Kabdasli, M. (2005). "Artificial intelligence methods in breakwater damage ratio estimation." *Ocean Engineering*, Elsevier, 32(17), 2088–2106.

Yang, C.-C. (2011). "Constructing a hybrid Kansei engineering system based on multiple affective responses: Application to product form design." *Computers & Industrial Engineering*, Elsevier, 60(4), 760–768.

Zhang, C., Shao, H., and Li, Yu. (2000). "Particle swarm optimisation for evolving artificial neural network." Systems, Man, and Cybernetics, 2000 IEEE International Conference on, IEEE, 4, 2487–2490.

Zhang, J., Zhan, Z.-H., Lin, Y., Chen, N., Gong, Y.-J., Zhong, J.-H., Chung, H. S., Li, Yun, and Shi, Y.-H. (2011). "Evolutionary computation meets machine learning: A survey." *Computational Intelligence Magazine, IEEE*, IEEE, 6(4), 68–75.

Zhou, H., Pei Zhao, J., Gang Zheng, L., Lin Wang, C., and Fa Cen, K. (2012). "Modeling NO x emissions from coal-fired utility boilers using support vector regression with ant colony optimization." *Engineering Applications of Artificial Intelligence*, Elsevier, 25(1), 147–158.

ABSTRACT

A PREDICTION MODELING FRAMEWORK FOR NOISY WELDING QUALITY DATA

by

JUNHEUNG PARK

August 2015

Advisor: Dr. Kyoung-Yun Kim

Major: Industrial Engineering

Degree: Doctor of Philosophy

Various research projects have been conducted to utilize historical manufacturing process data in product design. These manufacturing process data often contain data inconsistencies, and it causes challenges in extracting useful information from the data. In resistance spot welding (RSW), data inconsistency is a well-known issue. In general, such inconsistent data are treated as noise data and removed from the original dataset before conducting analyses or constructing prediction models. This may not be desirable for every design and manufacturing applications since every data can contain important information to further explain the process. In this research, we propose a prediction modeling framework, which employs bootstrap aggregating (bagging) with support vector regression (SVR) as the base learning algorithm to improve the prediction accuracy on such noisy data. Optimal hyper-parameters for SVR are selected by particle swarm optimization (PSO) with meta-modeling. Constructing bagging models require

more computational costs than a single model. Also, evolutionary computation algorithms, such as PSO, generally require a large number of candidate solution evaluations to achieve quality solutions. These two requirements greatly increase the overall computational cost in constructing effective bagging SVR models. Meta-modeling can be employed to reduce the computational cost when the fitness or constraints functions are associated with computationally expensive tasks or analyses. In our case, the objective function is associated with constructing bagging SVR models with candidate sets of hyper-parameters. Therefore, in regards to PSO, a large number of bagging SVR models have to be constructed and evaluated, which is computationally expensive. The meta-modeling approach, called MUGPSO, developed in this research assists PSO in evaluating these candidate solutions (i.e., sets of hyper-parameters). MUGPSO approximates the fitness function of candidate solutions. Through this method, the numbers of real fitness function evaluations (i.e., constructing bagging SVR models) are reduced, which also reduces the overall computational costs. Using the Meta^2 framework, one can expect an improvement in the prediction accuracy with reduced computational time. Experiments are conducted on three artificially generated noisy datasets and a real RSW quality dataset. The results indicate that Meta^2 is capable of providing promising solutions with noticeably reduced computational costs.

AUTOBIOGRAPHICAL STATEMENT

Junheung Park was born in Seongnam and raised in Seoul, South Korea. He completed his Bachelor's degree in Industrial Engineering in 2009 at Kyoungwon University. In 2011, he received his Master's degree in Computer Science at Wayne State University. Since then, he has been pursuing his PhD in Industrial Engineering at Wayne State University with a focus on machine learning and computational intelligence. He has applied these skills and experience to multiple industry and academic projects in product development, manufacturing, and data-driven system design.