CMES, vol.8, no.1, pp.61-72, 2005

Mining of Data from Evolutionary Algorithms for Improving Design Optimization

Y.S. Lian[1] and M.S. Liou[2]

Abstract: This paper focuses on integration of computational methods for design optimization based on data mining and knowledge discovery. We propose to use radial basis function neural networks to analyze the large database generated from evolutionary algorithms and to extract the cause-effect relationship, between the objective functions and the input design variables. The aim is to improve the optimization process by either reducing the computation cost or improving the optimal. Also, it is hoped to provide designers with the salient design pattern about the problem under consideration, from the physics-based simulations. The proposed technique is applied to both academic problems and real-world problems, including optimization of an airfoil and the turbopump of a cryogenic rocket engine. Our results demonstrate that these techniques can further improve the design already achieved by the evolutionary algorithms with a slightly additional cost.

Keywords: Evolutionary Computation, Data Mining, Optimization.

1 Introduction

Many real design problems in aerospace and aeronautical fields are often multi-objective, multi-modal and multidisciplinary in nature. Evolutionary algorithms (EAs) are particularly suitable for multi-objective optimization problems (MOOPs) because EA's population approach can be exploited to explore equally all non-dominated solutions in a population and keep an archive of a diverse set of non-dominated solutions. This striking feature gives EAs an advantage for MOOPs because they eliminate the need to convert a MOOP into multiple single-objective problems and the need to select different settings of parameters which unavoidably favor certain Pareto-optimal solutions [Deb (2001)].

Evolutionary computations usually produce enormous amount of data that contain rich information of which only a tiny fraction is generally utilized throughout the design optimization. Tremendous profit may be gained from the generated data if a systematic analysis of the data is performed so that a salient feature of the input-output relationship can be distilled. However, these data are not only large in volume but also highly multidimensional in space. Furthermore, unlike much of statistics, in which data are carefully chosen to answer specific questions, the data from EAs are usually collected randomly. Therefore, it is very difficult to analyze and root out distinctive features of the underlying physics.

Data mining is the analysis of large observable data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner [Hand et al.(2001)]. Data mining is a powerful new technology, it is an interdisciplinary field, adopting established algorithms from statistics, machine learning, neural networks, and databases. We propose to leverage the data mining techniques to further improve our evolutionary design optimization. In our approach, we first construct a model generalizing the relationship between the objective function and design variables. This is achieved using a radial basis function (RBF) neural network based on the available data. This model is then used to predict future input, i.e., we can achieve better results through optimizing the constructed model. Our work aims at (1) providing a robust, automatic, physics-based systematic approach to obtain global optimization to a complex engineering system, and (2) discovering the underlying causal relationships between objective functions and design variables at the end of the optimization. As a result, it can lead to a capability to yield a better design at a much reduced cost, by delegating the complex searching process to machine, while freeing scientists and engineers to focus on creative thinking/formulations. also, the second goal can lead to gaining succinct knowledge for similar problems by a systematic analysis of the enormous complex data set.

In the multidisciplinary optimization literature there is

[1] Ohio Aerospace Institute, Cleveland, OH 44142. Current address: University of Michigan, Ann Arbor, MI 48109
[2] NASA Glenn Research Center, MS 5-11, Cleveland, OH 44135

copious of work on using surrogate models such as response surface and Kriging to reduce the computational burden of optimizing complex engineering problems [Simpson et al. (2002), Lian, and Liou (2005)]. In the evolutionary computation surrogate models are also used to reduce the computational cost and to enhance convergence rate [Ong et al. (2003), Lian, and Liou (2004a, 2004b), Wang et al. (2004), Zhu et al. (2004)]. In those work surrogate models are constructed either before or during the optimization procedure. In the former one, surrogate models are constructed based on preselected design points, which are selected based on the design of experiment either to fill the space evenly or to concentrate on the boundary. Our work is different from those in that the model, which generalizes the relationship between the objective and design variables, is constructed after the evolutionary optimization is completed. There is no guarantee that those points are evenly distributed in the design space or representative of the entire space. More often their distribution is chaotic. And in evolutionary optimization, we expect that the data points are very close. Our mining of data is the post-processor of the optimization procedure. The aim is to extract useful information from the already existing data points.

This paper is constructed as follows: we first introduce the basic concept of RBF neural network, which includes ridge regression, bias-variance trade-off, basis function optimization, weight parameter determination, and data pre-processing. This is followed by numerical analysis. Four examples are presented to demonstrate the applicability and effectiveness of our approach, including design of a subsonic airfoil and design of a turbopump used in the space launch vehicle. Our results demonstrate that these techniques can expedite the design process, further improve the existing design. Without major changes, these technologies can also be extended to data mining in other fields.

2 Radial Basis Neural Networks

Neural networks simulate human functions such as learning from experience, extracting recognizable information or patterns from inputs containing irrelevant or even chaotic data. Neural computing has emerged as a practical technology and has been successfully applied in many fields such as time series analysis, pattern recognition, signal processing and control. Most of these works make use of feed-forward network architectures such as the multi-layer perceptron and the RBF neural networks. Here we adopt RBF neural networks to perform data mining because of their simplicity and ease to interpolate. RBF networks are a major class of neural network models, in which the activation of a hidden unit is determined by the distance between the input vector and a prototype vector. The RBF neural networks usually have the following form:

$$f(\vec{x}) = \sum_{j=1}^{m} w_j \phi_j(\vec{x}), \tag{1}$$

where $\vec{x}$ is an n-dimensional input vector with element x_i, $\{w_j\}_{j=1}^{m}$ are the weight parameters, and f is the model function expressed as a linear superposition of m basis functions $\{\phi_j\}_{j=1}^{m}$. Several forms of basis function have been considered, the most common being the Gaussian:

$$\phi_j(\vec{x}) = \exp(-\frac{||\vec{x}-\vec{\mu}_j||}{\sigma_j^2}). \tag{2}$$

Here $\vec{\mu}_j$ is the prototype vector determining the center of the basis function ϕ_j and it has elements μ_{ji}, σ_j is a parameter controlling the smoothness properties of the interpolating function. The Gaussian is a localized function with the property that $\vec{\phi}_j \rightarrow 0$ as $\vec{x}$ is far away from $\vec{\mu}_j$.

RBF networks usually consist of a two-stage training procedure. In the first stage, parameters governing the basis functions $\{m, \vec{\mu}_j, \sigma_j\}$ are determined. This is an unsupervised procedure, in which parameters are chosen to form an optimal representation of the probability density of the input data. The second stage determines the weight parameters $\{w_j\}_{j=1}^{m}$. Upon completion of the first stage, because basis functions are already determined, model function in Eq. (1) is simply a linear function of the weight parameters. If we have p training data, $\{\vec{x}^i, y_i\}_{i=1}^{p}$, $p \geq m$, then the weight parameters will be resolved by minimizing the sum-squared-error

$$E = \sum_{i=1}^{p} (y_i - f(\vec{x}^i))^2. \tag{3}$$

To restrict the flexibility of linear model (1), we usually augmented Eq.(3) with a penalty term. The function to be minimized then becomes:

$$E_g = \sum_{i=1}^{p} (y_i - f(\vec{x}^i))^2 + \lambda \sum_{i=1}^{m} w_i^2, \tag{4}$$

where $\lambda > 0$ is the regularization parameter. A small λ means great flexibility and leads to a tight fit without causing large penalty; a large λ favors the penalty term in the cost of flexibility and tight fit. The reason for using the regularization parameter and its effects will become clear in the following sections. The process of finding a proper λ is called weight decay in neural network field, or ridge regression in statistics field.

The second stage of RBF network training involves the determination of weight parameters. Depending on user's preference the weight parameters can be obtained by minimizing either Eq. (3) or Eq. (4), which leads to the following linear system of equations:

$$\vec{A}\vec{w} = \vec{H}^T\vec{y}, \tag{5}$$

where $\vec{w} = [w_1, w_2, \ldots, w_m]^T$ is the weight parameter vector, $\vec{y} = [y_1, y_2, \ldots, y_p]^T$ is the target information, and $\vec{H}$ is the design matrix with the following form:

$$\vec{H} = \begin{bmatrix} \phi_1(\vec{x}^1) & \phi_2(\vec{x}^1) & \ldots & \phi_m(\vec{x}^1) \\ \phi_1(\vec{x}^2) & \phi_2(\vec{x}^2) & \ldots & \phi_m(\vec{x}^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\vec{x}^p) & \phi_2(\vec{x}^p) & \ldots & \phi_m(\vec{x}^p) \end{bmatrix}. \tag{6}$$

$\vec{A}$ is the variance matrix. If Eq. (3) is used as the error function, then A has the following form:

$$\vec{A} = \vec{H}^T\vec{H}, \tag{7}$$

or if Eq. (4) is used, then the variance matrix $\vec{A}$ can be written as:

$$\vec{A} = \vec{H}^T\vec{H} + \vec{\Lambda}, \tag{8}$$

and $\vec{\Lambda}$ is a diagonal matrix,

$$\vec{\Lambda} = \begin{bmatrix} \lambda & & & \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \end{bmatrix}. \tag{9}$$

A useful matrix, which is used often in the RBF networks, is the projection matrix:

$$\vec{P} = \vec{I}_p - \vec{H}\vec{A}^{-1}\vec{H}^T. \tag{10}$$

This square matrix projects vectors in p-dimensional space perpendicular to the m-dimensional subspace spanned by the model. With a simple matrix operation we can have the following relation:

$$\vec{P}\vec{y} = \vec{y} - \vec{H}\vec{w} = \vec{y} - \vec{f}. \tag{11}$$

Eq.(11) tells us that $\vec{P}\vec{y}$ is the error vector due to the approximation of the real function $\vec{y}$ with model function $\vec{f}$ of Eq. (1).

With the matrix notation, the sum-squared-error in Eq. (3) can be equivalently written as:

$$E = \vec{y}^T\vec{P}^2\vec{y}, \tag{12}$$

or the error function in Eq. (4) can be written as:

$$E_g = \vec{y}^T\vec{P}\vec{y} \tag{13}$$

Unlike multi-layer perceptrons, RBF networks generally have a simple architecture consisting of two layers of weights, in which the first layer contains parameters of the basis functions (m, $\vec{\mu}_j$, and σ_j), and the second layer forms a linear combination of the activations of the basis functions to generate the outputs. Soon we will find that the training of RBF networks is substantially faster than the training of multi-layer networks, which is one of the principal advantages of the RBF networks.

2.1 *Bias-variance trade-off*

From our experience of polynomial fitting we know that a lower order polynomial fitting in general shows a poor approximation as a consequence of its limited flexibility while a high order polynomial can give rise to unwanted oscillations. Similarly, in the RBF network training of Eq. (1), there is an optimal number of basis functions $\{\phi_j\}_{j=1}^m$, which gives the best representation of the underlying systematic properties of the data, and hence obtains the best generation on new data. We can obtain a better appreciation of this phenomenon by introducing the concept of the bias-variance. Suppose the input is $\vec{x}$, then the trained model predicts the output as $f(\vec{x})$. If we had many training sets and we knew the true output $y(\vec{x})$, we could calculate the mean-squared-error as

$$\langle (y(\vec{x}) - f(\vec{x}))^2 \rangle \tag{14}$$

where $\langle . \rangle$ denotes the expectation over the training sets. For convenience, Eq. (14) can be decomposed into two components

$$\underbrace{(y(\vec{x}) - \langle f(\vec{x}) \rangle)^2}_{bias} + \underbrace{\langle (f(\vec{x}) - \langle f(\vec{x}) \rangle)^2 \rangle}_{variance}. \tag{15}$$

The first part is the bias and the second part is the variance. The bias measures the extent to which the average of the network function differs from the true value $y(\vec{x})$; the variance measures the extent to which the trained model $f(\vec{x})$ is sensitive to a particular choice of the data set.

An inflexible model will have a large bias while a flexible model will have a large variance. The best model is obtained when we have the best compromise between the conflicting requirements of small bias and small variance. In neural networks we can achieve such a compromise by adjusting the number of basis functions. Or, as an alternative, always referred as regularization in literatures, we can accomplish that by restricting the flexibility of linear model (1) using a penalty as shown in Eq.(4). In the following we describe both approaches.

2.2 Basis function optimization

The first stage of RBF training involves determination of the basis function parameters, i.e., the number of basis functions, m, basis function center, $\vec{\mu}_j$, and smooth parameter, σ_j. In general RBF network can approximate arbitrary non-linear functional mappings between multidimensional space. With Eq. (1) we can exactly map the whole input data directly onto the final outputs as long as the number of basis functions is equal to the number of training points. However, such an exact interpolation is typically a highly oscillatory function with large variance, generally it fails to give a good representation and usually produces poor results. In addition, our goal of network training is not to get an exact representation of the known input data but rather to make good representations for the unknown inputs. Therefore, it is necessary to choose the optimal settings of m, $\vec{\mu}_j$, and σ_j.

From the perspective of function approximation, generalization, and noisy interpolation, the basis function parameters should be chosen to form a representation of the probability density of the input data. A number of strategies have been proposed based on the following considerations. One straightforward approach is to randomly choose a number of subsets from the data, set the basis function centers $\vec{\mu}_j$ equal to the input vectors, and then chose the best candidate among those subsets. This approach is fast but clearly is not an optimal procedure insofar as the probability density estimation is concerned, and may also lead to the use of an unnecessarily large number of basis functions in order to achieve an adequate performance on the training data [Bishop (2003)].

A more sophisticated strategy is the forward selection procedure, which starts with an empty subset and adds one basis function each time. Suppose we have M training data points, forward selection starts with an empty subset, and M networks are trained in which each of the M data in turn is selected as the center for the basis function. The training point which most reduces the sum-squared-error (Eq.(3)) is retained. In subsequent steps of the algorithm, the number of basis functions is then increased incrementally. This process stops until some chosen criteria is satisfied. Forward selection involves growing the network by one basis function.

Suppose we add a new basis function, ϕ_J, to the subset which already has m basis functions, then the new design matrix in Eq. (6) is

$$\vec{H}_{m+1} = [\vec{H}_m \;\; \vec{h}_J], \tag{16}$$

where $\vec{H}_m$ is the old design matrix with m basis functions, and $\vec{h}_J$ is

$$\vec{h}_J = \begin{bmatrix} \phi_J(\vec{x}^1) \\ \phi_J(\vec{x}^2) \\ \vdots \\ \phi_J(\vec{x}^p) \end{bmatrix}. \tag{17}$$

The newly added basis function is the one that reduces the sum-squared-error the most. Without calculating the weight parameters from scratch with Eq. (5), we can greatly reduce the cost by using appropriate incremental operation [Orr (1996)]. The key equation is

$$\vec{P}_{m+1} = \vec{P}_m - \frac{\vec{P}_m\vec{h}_J\vec{h}_J^T\vec{P}_m}{\vec{h}_J^T\vec{P}_m\vec{h}_J}, \tag{18}$$

where $\vec{P}_m$ is the projection matrix in Eq. (11) when there are m basis functions in the subset; $\vec{P}_{m+1}$ is the subsequent projection matrix when one more basis function is added. Two RBF models are constructed, one has m basis functions, and the other has $m+1$. With Eq.(3), the training errors, associated with the two models and evaluated with the p available data, can be written as follows:

$$E_m = \vec{y}^T\vec{P}_m^2\vec{y}, \tag{19}$$
$$E_{m+1} = \vec{y}^T\vec{P}_{m+1}^2\vec{y}. \tag{20}$$

Therefore, adding a new basis function will reduce the sum-squared-error in an amount of

$$\hat{\sigma}^2 = E_m - E_{m+1} = \frac{(\vec{y}^T\vec{P}_m\vec{h}_J)^2}{\vec{h}_J^T\vec{P}_m\vec{h}_J}. \tag{21}$$

The approximate number of multiplications required to add a basis function is $2p^2$, which is much less than the cost of simply calculating a projection matrix from scratch, which is about $m^3 + pm^2 + p^2m$. This process can be further improved with the orthogonal least square technique [Chen et al. (1989)], which has been implemented by Orr (1996, 1999) in his Matlab package for RBF networks and is used in our study. It is noted that for the RBF network, we can choose suitable parameters for the hidden units without having to perform a full non-linear optimization problem as required by the multi-layer perceptron networks.

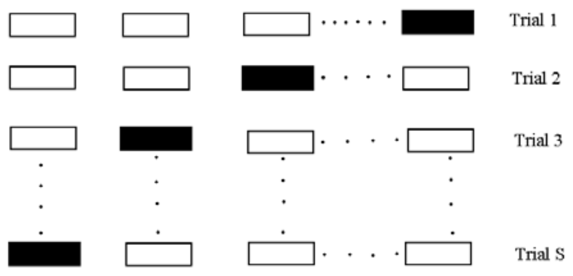

Figure 1 : Illustration of the hold-out method of cross-validation. For a given trial, the shaded subset of data is used to validate the model trained on the remaining data.

2.3 Generalized cross-validation

In the basis function optimization we grow the subset by one basis function at a time until the sum-squared-error has been reduced the most. In this choosing process, we compute the sum-squared-error with Eq. (3) by substituting the known data $\{\vec{x}^i, y_i\}_{i=1}^p$. Ordinarily, this sum-squared-error will decrease as extra functions are added: it starts off at a large value, deceases rapidly, and then continues to decrease slowly as the network makes its way to a local minimum on the error surface. If continuing the process further, it leads to an exact interpolation and ends up overfitting the training data. However, our goal in optimizing the basis function is not to have an exact interpolation of the known data but to have a well-trained model which has the least prediction error for the unknowns. Unfortunately, for an arbitrary input $\vec{x}$, the true output $y(\vec{x})$ is usually unknown. Therefore, it is required to select a proper model to estimate the prediction error. Many popular models for prediction error have been attempted.

In this context, cross-validation provides an appealing guiding principle [Stone (1974)]. The philosophy of cross-validation is to randomly partition the available data into a training set and a test set. Different models trained on the training set can be compared on the test set. To guard against the possible bias associated with a particular partition, the available data set can be divided into S distinct segments and an average score over the partitions is evaluated. A network is trained using data from the $S-1$ segments, and its performance is evaluated by computing the error function using the remaining segment as illustrated in Fig. 1. This process repeats for all the S possible choices. The performance of the model is assessed by averaging over all the errors. Consequently, it avoids the bias that may be introduced by an arbitrary division of the data. In addition, all the data can be used for training with a high proportion of the available data. When the available data is limited, we may go to the extreme limit of $S = p$ for a data set with p data. This limit is known as the leave-one-out method. The generalized cross-validation (GCV) by Golub et al. (1979) is an improved version of cross-validation with the same underlying idea. The beauty of GCV for the linear RBF networks is further illustrated by an analytical expression for the average variance over the training set:

$$\hat{\sigma}^2 = \frac{p\vec{y}^T\vec{P}^2\vec{y}}{(\mathrm{trace}(\vec{P}))^2} \tag{22}$$

We can interpret the GCV as a good approximation to the prediction error when $p-1$ basis functions are used to construct the model. In addition, it is noted that the penalty parameter is absent in the evaluation of the GCV error and $\mathrm{trace}(\vec{P}) = p - m$.

From Eq. (21), we get $E_m - E_{m+1} \geq 0$, which means that adding a basis function will always decrease the training error. Training error is estimated based on a subset, on which the model design is based. It is possible for the network to end up overfitting the training data if the training section is not stopped at the right point. GCV, on the other hand, is validated based on subset, which is independent from the model design, and will stop decreasing error as a consequence of overfitting. Therefore, we adopt GCV to monitor our training process. We terminate the forward selection process once GCV stop decreasing.

2.4 Ridge regression

The regularization parameter λ in Eq.(4) restricts the flexibility of the linear model by introducing a small amount of bias. The optimal value of λ is associated with the least prediction error. By using GCV, the optimal value of λ can be found with the following formula:

$$\lambda = \frac{\vec{y}^T \vec{P}^2 \vec{y} \mathrm{trace}(\vec{A}^{-1} - \lambda \vec{A}^{-2})}{\vec{w}^T \vec{A}^{-1} \vec{w} \mathrm{trace}(\vec{P})}, \tag{23}$$

where $\vec{w}$ is a vector satisfying Eq.(5). Equation (23)is an implicit equation for λ and a numerical iteration procedure must be used until the solution is converged.

2.5 Data Pre-processing

Hartman et al. (1990) proved that Eq. (1) is capable of universal approximation, i.e., RBF networks can perform an arbitrary non-linear functional mapping. However, such a mapping usually produces poor results. For example, in the RBF networks, it is particularly important to normalize the input variables so that they span similar ranges. For example, Gaussian function in the RBF network is determined by the Euclidean distance between the input $\vec{x}$ and the prototype $\vec{\mu}_j$. If one of the input variables has a much smaller range of values than the others, then the distance (Gaussian function) will be insensitive to variation of this variable. One approach is to apply a linear transformation to the inputs so that they all have the same ranges. This linear transformation can be based on the variable variance or the maximal/minimal value. If it is based on the variance, the scaled new variable is:

$$\tilde{x}_i^j = \frac{x_i^j - \bar{x}_i}{\sigma_i}, \tag{24}$$

where $\{x_i^j\}_{j=1}^p$ are p input variables, $\bar{x}_i$ and σ_i are the mean and variance of variable x_i, respectively, whose definitions are as follows:

$$\bar{x}_i = \frac{1}{p} \sum_{j=1}^{p} x_i^j, \tag{25}$$

Here we treat variables separately. With this transformation the scaled variable $\tilde{x}_i^j$ then has a zero mean and unit variance. Or we can scale the source data to a space $[0\ 1]^n$ based on their maximal/minimal value using a liner transformation. In the RBF network package by Orr (1999) the source data is equivalently transformed to a space $[0\ 1]^n$ by using different smooth parameters in different dimensions. Unlike Eq.(2), the basis function now has the following form:

$$\phi_j(\vec{x}) = \exp(\sum_{i=1}^{n} \frac{(x_i - \mu_{ij})^2}{\sigma_{ji}^2}), \tag{26}$$

where σ_{ji} is the smooth parameter in the i-th dimension. It is proportional to the range of value in that dimension.

3 Data mining process

Our data mining process begins with data pre-processing of the raw data. In this stage we may remove those repetitive data, filter the obvious noisy data, and sort the data. We can also transform the data to have the same range of values. After pre-processing, we construct the RBF network with a number of training data. The constructed model function is then optimized with an optimizer. The optimal solutions from the model function are usually not the true optimal solution of the real function, hence they need to be further validated based on the real function.

4 Numerical Analysis

In this section four examples are tested with our proposed approach. Two of them are unconstraint optimization problems widely used in the literature; the other two are real engineering problems with constraints. The raw data are from the design optimization with a real-coded GA [Arakawa, and Hagiwara (1997)]. In the GA, a blend crossover (BLX-α) operator is used with a value of α=0.5. In our computations, all design variables are scaled to the range of [0 1]. We choose a uniform mutation operator which adds a uniform random number to the parent solution at a probability of p_c:

$$y_i = x_i + (r_i - 0.5)\Delta_i, \tag{27}$$

where r_i is a random number, Δ_i is a user-defined maximum perturbation allowed in the i-th decision variable. We set p_c=0.1 and $\Delta_i = 0.1$ in our computations. To ensure a monotonic improvement for the GA, we adopt the elitist strategy [De Jong (1975)] in which some of the best individuals are copied directly into the next generation without applying any evolutionary operators. For single objective optimization, sorting is based on the value of the objective function; for multi-objective optimization, we rank solutions based on Goldberg's non-dominated sorting procedure [Srinivas, and Deb (1994)].

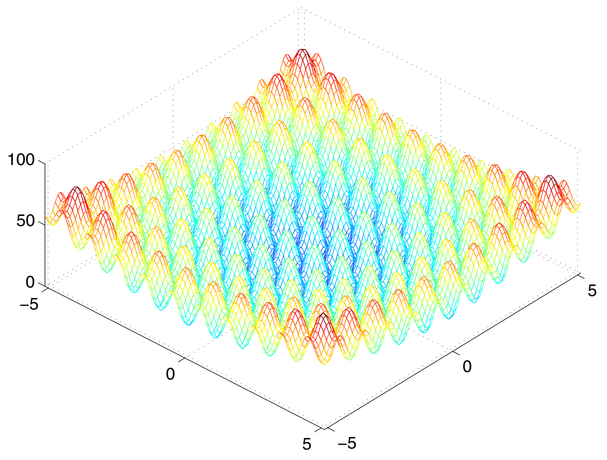

Figure 2 : Plot of two-dimensional Rastrigin function.

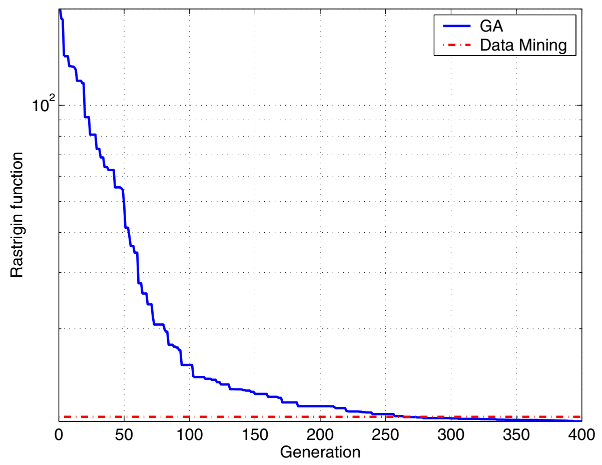

Figure 3 : Convergence history of Rastrigin function.

To maintain a uniform distribution on the Pareto-optimal front, we use fitness sharing [Goldberg et al.(1989)] in the multi-objective optimization. The optimization on the constructed model is performed with a sequential quadratic programming solver available in commercial optimizer DOT.

First we apply our approach on the Rastrigin problem, which is commonly used in the global optimization literature. The Rastrigin problem is defined as follows:

$$\begin{array}{ll} \text{Minimize:} & \\ & 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)], \\ \text{Subject:} & \\ & -5.12 \leq x_i \leq 5.12, \quad i = 1, 2, \ldots, n. \end{array} \tag{28}$$

The test problem has a highly bumpy surface with many local optima, which is illustrated from in Fig. 2 in which $n = 2$. In our test we set $n = 20$. Needless to say, this is a challenging high-dimension problem. The function has a global minimal value of zero at $x_i = 0$. We set the population size of 100 in the genetic computation. Fig. 3 shows the convergence history of the optimization process. The convergence rate slows down after 100 generations; further computations improve the optimal value but the improvement is insignificant. Therefore, we terminate our optimization process at the 100-th generation and switch to data mining aiming at improving the existing optimal solution. At the 100-th generation, the minimal function value is 15.4106.

With 100 generations of computation we generate 10,000 data. If we directly use those data to train the neural network, the predictive results will be poor partially because of the curse of dimensionality. By removing the repetitive data we have 2,092 data left. We then sort the remaining data in ascending order based on the function value. The first p points are chosen as the training data. Ideally, the selected training points should accurately represent the density function. However, there is no theoretical analysis on how many training points should be used in the neural network training for an optimal fitting. Furthermore, unlike those generated from the design of experiment, our data are not collected using efficient strategies for data analysis, instead they are generated from evolutionary computations.

We test different sets of training points. Their effect on data mining performance is illustrated in Table 1. With 100 training points, the RBF network is constructed with 98 basis functions. We use software DOT to find the optimal solutions of the constructed network. We call these solutions "constructed optimal solutions". Optimization with DOT on the model function predicts the same minimal as the genetic computation, indicating the constructed network gives a good representation of the real function. The "constructed optimal solutions" are verified against the real function. We obtain a minimal of 11.7623, a value can be achieved with 82 more generations of computation using genetic algorithm.

With the increase of training points we still get improvement but it is not as good as this one. With 600 and 800 training points, no improvement can be made any more and in fact the result begin to diverge from the baseline. There are two potential reasons behind this phenomenon. From Table 1 we can see the number of basis functions increases with the number of training points. Basis func-

Table 1 : Effect of number of training points on data mining performance for Rastrigin function.

Input	Basis functions	DOT	Verified
100	98	15.4106	11.7623
200	198	15.4106	12.0066
300	216	11.2107	13.0483
400	298	11.1723	12.6815
500	492	15.3712	13.6413
600	495	10.8375	41.8475
800	718	11.1086	208.9683

tion gives the freedom to fit the training data. More basis functions means less bias at an expense of more variance, thereby giving a poorer representation of the real function. Another reason is that the increase of training points is insufficiently rapid in relation to the domain of function for which a model can account. Because our neural network is constructed within a domain determined by the maximal and minimal values of the training points, the model function accuracy will deteriorate as a disproportional increase of domain and training points occurs. Our optimizer will take advantage of this inaccuracy and give false optimal solutions. Suppose the domain expanded by training database is a hypersphere with radius R, which we can define as follows:

$$R = \frac{1}{2}\sqrt{\sum_{i=1}^{n}(x_i^U - x_i^L)^2}. \tag{29}$$

Here, x_i^U and x_i^L are the maximal and minimal values of the i-th design variable amongst the m training data, respectively. If the hypersphere is equally divided into m small hypersphere with a radius of r, then we have

$$m(cr^n) = cR^n. \tag{30}$$

Thus, radius r can be expressed as the following:

$$r = \frac{\sqrt{\sum_{i=1}^{n}(x_i^U - x_i^L)^2}}{2\sqrt[n]{m}} \tag{31}$$

We plot radius versus number of training points in Fig. 4. In the figure we normalize the radius with r_0, the radius when $m = 100$. Figure 4 shows that radius increases with the number of training points, which means that the training points become more sparse in the domain. The sparsity reduces the accuracy of the model function. The radius initially shows a linear relation with the number of

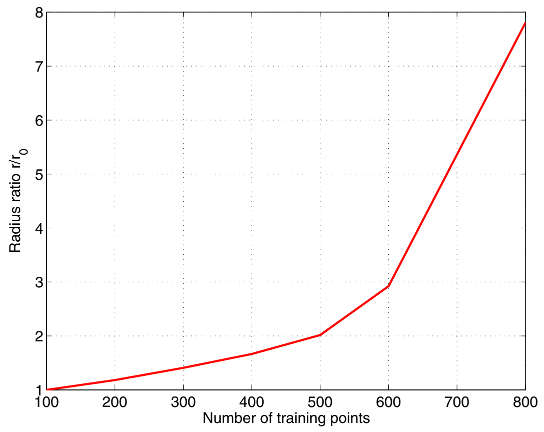

Figure 4 : Number of training points versus the radius for Rastrigin problem.

training points, then there is a dramatic increase when the number of training points is larger than 600. This increase tells us that there are not enough training points to construct an accurate representation of the real function.

The second problem we test is the Griewank function defined as

$$f(\vec{x}) = \sum_{i=1}^{n}\frac{x_i^2}{50} - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1. \tag{32}$$

This function has a global minimum zero locating at $(x_1, ..., x_n) = (0, ..., 0)$. We consider the optimization of the Griewank function with $n = 50$. The search domain is defined by $[-20\ 20]^{50}$. We set population size as 200. After 20,000 function evaluations, the minimal at the 100-th generation is 0.24209. The convergence history is shown in Fig. 5. We take these data as the training set to conduct data mining with the RBF neural network. Data mining results with different numbers of training points are shown in Table 2. Among them, the best result is obtained with 400 training points. Fig. 5 shows the convergence history for GA. As we can see from this convergence history, it takes GA 150 more generations to match the result from data mining. From the radius ratio shown in Fig. 6 we know that the variation from 100 training points to 800 training points is relatively modest, which means the training points are clustered closely. As a consequence of that, we still get improvement with data mining when the training point number is 800.

We now apply data mining to an airfoil design optimization. The objective is to design an airfoil with the maximal lift-to-drag ratio when the freestream Mach num-

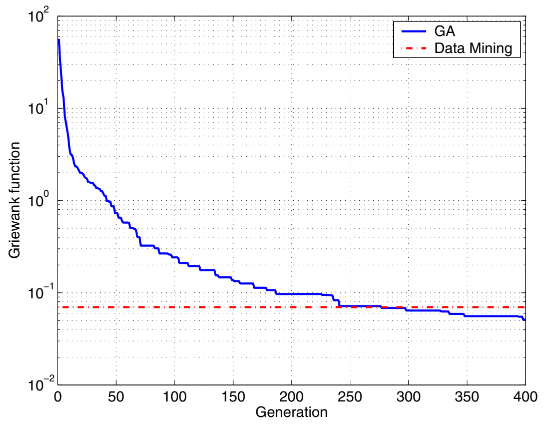

Figure 5 : Convergence history of the Griewank function.

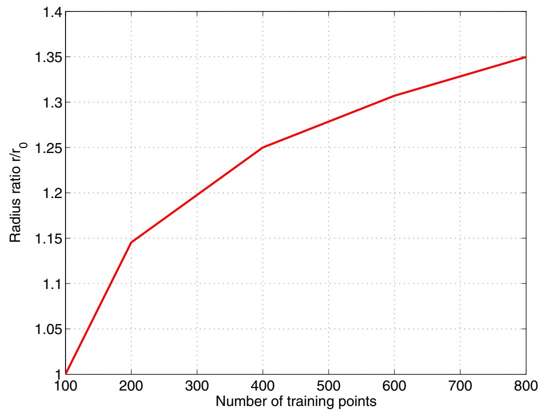

Figure 6 : Number of training points versus the radius for Griewank problem.

Table 2 : Effect of number of training points on data mining performance for Griewank function.

Input	Basis functions	DOT	Verified
100	53	0.16435	0.10587
200	192	0.24209	0.07879
400	398	0.24209	0.06973
600	598	0.24209	0.07463
800	798	0.24209	0.07980

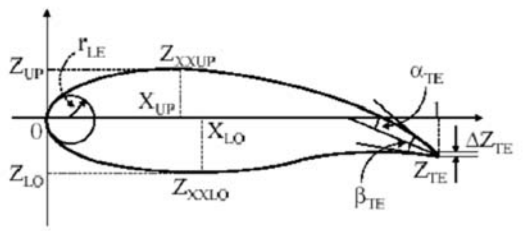

Figure 7 : Airfoil geometry defined by 11 basic parameters.

ber and the angle of attack are set to 0.70 and 2 degree, respectively. The airfoil thickness is constrained to be greater than 9.8% of the chord length at 60% of the chord. The airfoil is defined by the Sobieczky shape function [Sobieczky (1998)], which uses basic geometric parameters as design variables. The 11 design variables are shown in Fig. 7: leading-edge radius, upper and lower crest location including curvatures, trailing-edge coordinate, thickness, direction, and wedge angle. The aerodynamic performance of each design is evaluated with a two-dimensional Navier-Stokes solver based on a TVD-type upwind scheme [Obayashi, and Wada (1994)] with multigrid approach. In this study, both the trailing-edge coordinate and thickness are frozen to zero, which leaves nine design variables in our study. The real-coded adaptive-range GA is used to maximize the objective function where the population size and the number of generations are 64 and 100, respectively.

The maximal lift-to-drag ratio from the GA is 56.1674 at the 100-th generation. Eliminating the repetitive ones, 6,222 solutions are left. These data are sorted in descending order. We train the network with the first 100 data, among which 79 centers are chosen to built the neural network. The optimization with the DOT on the constructed network give the maximum of 56.1674, which is identical to the solution from GA. The best 20 "constructed optimal solutions" are validated by solving the Navier-Stokes equations. Upon completion of validation, the best solution we have is 56.1696. As shown in table 3 this result can be further improved by increasing the number of training data. With 400 training data, we built a neural network with 368 basis functions and we get an improved solution of 56.1706. However, the optimal solution with 800 training data is not as good as that with 400 training data. Fig. 8 shows that the radius increases with the number of training data. Although we expect model with large radius ratio is less accurate than that with small ratio, we do not have conclusion why the solution with $m = 400$ is better than the solution with $m = 100$.

The last problem we study is the design of a single-stage centrifugal pump. This problem has been studied by

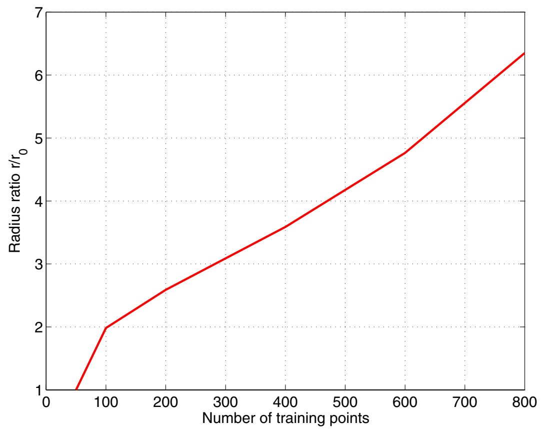

Figure 8 : Number of training points versus the design space radius for airfoil design.

Table 3 : Effect of number of training points on data mining performance for airfoil optimization.

Input	Basis functions	DOT	Verified
100	79	56.1674	56.1696
200	175	56.1674	56.1696
400	368	56.1674	56.1706
600	565	56.1674	56.1706
800	764	56.1674	56.1692

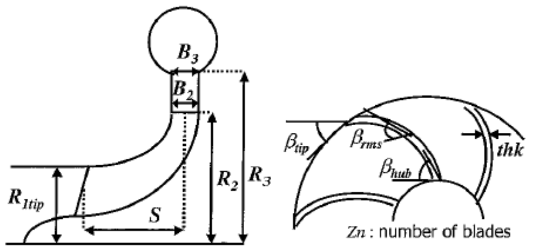

Figure 9 : Design parameters of the centrifugal pump design problem. (From Oyama and Liou (2002))

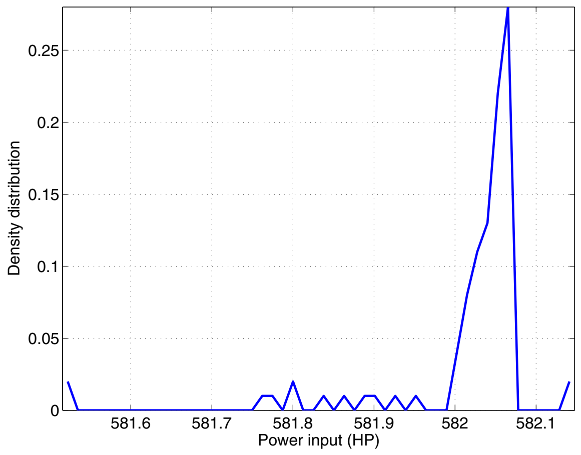

Figure 10 : Probability density function of the training points for turbopump problem.

Oyama and Liou (2003) with the genetic algorithm as a multiobjective optimization design, later this problem is further studied by Lian and Liou (2004b) with an enhanced genetic algorithm. The objective is to minimize the input power with the fixed head at a design point. The baseline design has a shaft rotating speed of 5,416.7 rpm, total temperature of the fluid entering the pump of 29.6°C, total pressure of the fluid entering the pump of 35,200kgf/m^2, and a mass flow rate of 85.67 kg/s. At the design point the head rise is 1045.9 inch and required input is 592.2 HP.

In this single-stage pump design, there are 11 design variables. They are the rotor leading-edge tip radius R_{1tip}, rotor trailing-edge radius R_2, volute tongue radius R_3, blade span at trailing edge B_2, blade span at volute tongue B_3, axial length of the blade at the rms diameter S, number of blades Z_n, blade thickness thk, blade trailing-edge angle at the hub, rms radius, and tip (β_{hub}, β_{mid}, β_{tip}), as shown in Fig. 9. The design spaces are listed in Table 4. Total head and required power input of pump design candidates are evaluated by using the one-dimensional mean-line pump flow modeling method [Veres (1994)].

The real-coded GA is applied. The population size and number of generations are set to 110 and 100, respectively. With the GA, we are able to reduce the power input from the original 592.2 HP to 581.5 HP. To perform data mining, we need to construct two neural networks, one is for the objective function, and the other for the constraint. Both are constructed with the same training database. However, the number of basis functions may be different. The computed results are tabulated in Table 5. In all cases, even though no further improvement is gained with data mining, we are still able to recover the results from GA. We plot the probability density function as a function of power input in Fig. 10. Most of the training points are clustered near 582.07, only a small portion of them are located near the minimal value of 581.5. This indicates that the model function is inaccurate in the sparse region, resulting in an ineffective performance.

Table 4 : Design parameter spaces for the centrifugal pump design problem.

Design Variables	$R_{1,tip}$ in.	R_2, in.	R_3, in.	B_2, in.	B_3, in.	
Lower bound	3.40	5.00	5.60	0.70	0.85	
Upper bound	4.00	5.60	6.20	0.85	1.00	
Design Variables	S, in.	β_{hub}, Deg.	β_{rms}, Deg.	β_{tip}, Deg.	thk, in	Z_{n2}
Lower bound	3.70	25.0	25.0	25.0	0.03	4
Upper bound	4.30	45.0	45.0	45.0	0.10	30

Table 5 : Effect of number of training points on data mining performance for airfoil optimization.

Input	Basis functions		DOT		Verified	
	Objective	Constraint	Objective	Constraint	Objective	Constraint
100	95	95	581.5	1045.0	581.5	1045.0
200	196	192	581.5	1045.0	581.5	1045.0
400	390	386	581.5	1045.0	581.5	1045.0

5 Conclusion

We performed the data mining for design optimization based on the available data generated from the evolutionary algorithms. This was achieved by constructing a RBF network, which was then optimized with a gradient-based optimizer. The performance was found to be problem dependent. Both the number of training points and training point distribution were important in determining the overall performance. Among the four problems presented, two unconstrained and two constrained optimizations, we have achieved further improvement in three of them, with only a fraction of one EA generation, in relation to the optimized results terminated by the GA procedure. In the fourth problem, we could recover the result from the evolutionary optimization, but no improvement was realized.

Acknowledgement: This work is supported by NASA research grand NAG3-2869 under the Ultra Efficient Engine Technology Program.

References

Arakawa, M., and Hagiwara, H. (1997): Nonlinear integer, discrete and continuous optimization using adaptive range genetic algorithms. *Proceedings of the 34th Design Automation Confernece*, Anaheim, CA.

Bishop, C.M. (2003): *Neural Networks for Pattern Recognition*, New York, Oxford University Press Inc.

Chen, S., Billings, S.A., and Luo, W. (1989): Orthogonal Least Squares Methods and Their Application to Non-linear System Indentification. *International Journal of Control*, Vol. 50, pp.1873–1896.

Deb, K. (2001): *Milti-Objective Optimization using Evolutionary Algorithm*, John Wiley & Sons, Chichester.

De Jong, K.A. (1975): An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D dissertation, University of Michigan, Ann Arbor.

1995, *DOT User's Manual. Version 4.20*. Vanderplaats Research & Development, Inc., Colorado Springs, CO.

Goldberg, D.E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Addison-Wesley.

Golub, G.H., Heath, M., and Wahba, G. (1979): Generalised Cross-validation as a Method for Choosing a Good Ridge Parameter. *Technometrics*, Vol. 21, pp.215–223.

Hand, D., Mannila, H., and Smyth, P. (2001): *Principles of Data Mining*, Cambridge, the MIT Press.

Hartman, E.J., Keeler, J.D., and Kowalski, J.M. (1990): Layered Neural Networks with Gaussian Hidden Units as Universal Approximations. *Neural Computation*, Vol. 2, pp.210–215.

Lian, Y., and Liou, M.S. (2004): Multiobjective Optimization Using Coupled Response Surface Model and

Evolutionary Algorithm. Accepted for publication at *AIAA Journal*. Also AIAA Paper 2004-4323.

Lian, Y., Liou, M.S., and Oyama, A. (2004): An Enhanced Evolutionary Algorithm with a Surrogate Model. Genetic and Evolutionary Computation Conference, Seattle, WA, 2004.

Lian, Y., and Liou, M.S. (2005): Multiobjective Optimization of a Transonic Compressor Rotor using Evolutionary Algorithm. Accepted for publication at *Journal of Propulsion and Power*. Also AIAA Paper 2005-1816, 2005.

Obayashi, S., and Wada, Y. (1994): Practical Formulation of a Positively Conservative Scheme. *AIAA Journal*, Vol. 32, pp.1093–1095.

Ong, Y.S., Nair, P.B., and Kean, A.J. (2003): Evolutionary Optimization of COmputationally Expensive Problems via Surrogate Modelling. *AIAA Journal*, Vol. 41, pp.687–696.

Orr, M.J.L. (1995): Regularisation in the Selection of Radial Basis Function Centers. *Neural Computation*, Vol. 7, pp.606–623.

Orr, M.J.L. (1999): Introduction to Radial Basis Function Networks. http://www.anc.ed.ac.uk/ mjo/

Oyama, A., and Liou, M.S. (2002): Multiobjective Optimization of Rocket Engine Pumps Using Evolution Algorithm. *Journal of Propulsion and Power*, Vol. 18, pp.528–535.

Simpson, T.W., Booker, A.J., Ghosh. D., Giunta, A.A., Koch, P.N., and Yang, R.J., (2002): Approximation Methods in Multidisciplinary Analysis and Optimization: A Panel Discussion. *Proceeding of the Third ISSMO/AIAA Internet Conference on Approximation in Optimization*, October, 2002.

Sobieczky, H. (1998): Parametric Airfoils and Wings. In: Fujii, K., and Dulikravich, G.S. (eds) *Notes on Numerical Fluid Mechanics*, Vol. 68, Vieweg Verlag, pp.71–88.

Srinivas, N., and Deb, K. (1994): Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, Vol. 12, pp.221–248.

Stone, M. (1974): Cross-validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society*, Vol. 9, pp.127–139.

Veres, J. P. (1994): Centrifugal and Axial Pump Design and Off-design Performance Prediction, NASA TM-106745.

Wang, M. Y., and Zhou, S.(2004):Phase Field: A Variational Method for Structural Topology Optimization, *CMES:Computer Modeling in Engineering & Sciences*, Vol. 6, No. 6, pp. 547-566.

Zhu, Z. Q., Liu, Z., Wang, X. L., and Yu, R. X.(2004): Construction of Integral Objective Function/Fitness Function of Multi-Objective/Multi-Disciplinary Optimization, *CMES: Computer Modeling in Engineering & Sciences*, Vol. 6, No. 6, pp. 567-576.