# Computationally Effective Search and Optimization Procedure Using Coarse to Fine Approximations

**Pawan K. S. Nain and Kalyanmoy Deb**
Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
{pksnain, deb}@iitk.ac.in
http://www.iitk.ac.in/kangal

**Abstract- This paper presents a concept of combining genetic algorithms (GAs) with an approximate evaluation technique to achieve a computationally effective search and optimization procedure. The major objective of this work is to enable the use of GAs on computationally expensive problems, while retaining their basic robust search capabilities. Starting with a coarse approximation model of the problem, GAs successively use finer models, thereby allowing the proposed algorithm to find the optimal or a near-optimal solution of computationally expensive problems faster. A general methodology is proposed for combining any approximating technique with GA. The proposed methodology is also tested in conjunction with one particular approximating technique, namely the artificial neural network, on a B-spline curve fitting problem successfully. Savings in the exact function evaluation upto 32% are achieved. The computational advantage demonstrated here should encourage the use of the proposed approach to more complex and computationally demanding real-world problems.**

## 1 Introduction

One of the main hurdles faced by an optimization algorithm in solving real-world problems is their need of a reasonably large computational time in finding an optimal and a near-optimal solution. In order to reduce the overall computational time, researchers in the area of search and optimization look for efficient algorithms which demand only a few function evaluations to arrive at a near-optimal solution. Although successes in this direction have been achieved by using new and unorthodox techniques (such as evolutionary algorithms, tabu search, simulated annealing etc.) involving problem-specific operators, such techniques still demand a considerable amount of simulation time, particularly in solving computationally expensive problems. In such problems, the main difficulty arises due to large computational time required in evaluating a solution. This is because such problems either involve many decision variables or a computationally involved evaluation procedure, such as the use of finite element procedure or a network flow computation.

Although the use of a parallel computer is a remedy to these problems in reducing the overall computational time, in this paper, we suggest a fundamental algorithmic change to the usual optimization procedure which can be used either serially or parallely. Most search and optimization algorithms begin their search from one or more random guess solutions. Thus, the main task of a search algorithm in the initial few iterations is to provide a direction towards the optimal region in the search space. To achieve such a task, it may not be necessary to use an exact (or a very fine-grained) model of the optimization problem early on. An approximate model of the problem may be adequate to provide a reasonably good search direction. However, as the iterations progress, finer models can be used successively to converge closer to the true optimum of the actual problem. Although this idea of using an approximate model in the beginning of a search algorithm and refining the model with iterations is not new [1, 6, 9, 11, 12], we suggest a generic procedure which can be used in any arbitrary problem.

In the reminder of this paper, we describe the proposed coarse-to-fine grained modeling procedure. Thereafter, we suggest an artificial neural network (ANN) based procedure, specifically to model an approximate version of the actual problem and show simulation results of the proposed technique applied to a two-objective geometric design problem. Different variations of the ANN design and training procedures are compared. Simulation results show a large computational advantage of the proposed procedure, thereby suggesting the applicability of the proposed procedure in more complex real-world search and optimization problems.

## 2 Coarse to Fine Grained Methods Used in Past Studies

The use of coarse-to-fine grained modeling in optimization can be found in various application papers available in the literature, particularly in computational fluid dynamics applications and complex mechanical component design problems. One such application is the optimal design of elastic flywheels using the injection island GA (iiGA) suggested by Eby et. el. [6]. It uses a finite element code to assist the iiGA to evaluate the solutions to find the specific energy density of flywheels. Similar work using the hierarchical genetic algorithm (HGA) for a computational fluid dynamics problem is reported by Sefrioui et. el. [11]. They used a multi-layered hierarchical topology to solve a classical exploration/exploitation dilemma while using multiple models for optimization problems. They have reported to achieve the same quality results as that obtained by a simple GA, but spending only about one-third of the computational time. The other important work in this area is reported by Poloni et.el. [12]. They have developed a methodology which uses a multi-objective genetic algorithm (MOGA) for

exploring the design space to find a local Pareto-optimal set of solutions. Next, they train a neural network with the database of solutions obtained so far to get the global approximation of the objective function. Finally, by defining the proper weights to combine the objectives, a single objective optimizer using the conjugate gradient method is run on the globally approximated objective function obtained earlier. They tested this methodology for the design of sailing yacht fin keel problem, coupling their optimization code to a 3D Navier-Stokes equation solver. A recent work combines the approximating concept with GA for structural optimization applications [13]. Investigators have tried to reduce the number of exact computations while ensuring to converge to the optima of the original problem. They claimed to reduce the exact analysis by more than 97% for the optimal design of a 10-bar truss structure. Despite the success, it remains to be seen what advantage will be achieved in a more and realistic truss structure optimization problem. Jin and Sendhoff [9] provide a good survey of approximation methods, while Branke and Schmidt [1] have recently reported a faster convergence achieved using local models based on interpolation and regression techniques.

## 3 Proposed Approach

We propose to combine a GA with the approximation technique which allows a GA to require a reduced number of function evaluations in solving computationally expensive problems. Since the function evaluation of solution vectors is required at every generation for fitness assignment in the GA procedure and for most of the practical applications GA is usually run for hundreds, if not thousands, of generations with a significant population size, it turns out to be the most desirable place for improving the computational efficiency of a traditional GA procedure. In order to reduce the computational time required to execute one function evaluation, the following strategies can be used:

- Use a partial evaluation of a solution
- Use a parallel computer
- Use an approximation of the optimization problem

Certain search and optimization problems may be functionally decomposable into a number of subproblems. In such problems, the most important subproblems can only be evaluated in the initial GA generations. Although this procedure will introduce some error in evaluating a solution in early generations (since not all subproblems are evaluated), the computations can be performed quickly. As mentioned earlier, in the early generations the task of an optimizer is to determine correct search directions towards the optimum, such errors may not cause a large deviation from the true search direction. However, as generations progress, more and more less important subproblems can be included and more accurate function evaluations are expected.

Because of the availability of parallel computers, it may be plausible to take advantage of parallel computing of different tasks involved in a function evaluation. For example, to evaluate a solution involving FFT or finite element computations, the solution can be sent to multiple processors for a faster computation. Since GAs use a population of solutions in each generation, most parallel GA applications perform a distributed computing of allocating a complete solution to each available processor, thereby reducing the overall computational time to complete one generation.

The focus of this study is to use a successive approximation of the optimization problem. Starting with a coarsely approximated model of the problem, GAs use successively fine-grained models with generations. Figure 1 depicts this procedure. The figure shows a hypothetical one-dimensional objective function for minimization in a solid line. Since this problem has a number of local minimum solutions (which is one of the difficulties often exist in a real-world optimization problem), it would be a difficult problem for any optimization technique. It is concluded elsewhere [7] that to find the global optimum in such a problem using a GA, a population of size $O(\gamma^2)$, where $\gamma$ is the inverse of the signal-to-noise of the function, is needed. The signal being the difference between the global and the next-best local optimal values and the noise being equal to the variance of the function values. Thus, the objective function shown in the figure demands a large population size, if the GA has to start from an initial random population. Figure 1 shows a coarsely approximated function in the entire range of the function with a dashed line. There could be a variety of ways such an approximated function can be obtained:

- Linear or quadratic approximation of the true function
- Approximation through a set of basis functions
- Approximation through a chosen set of solutions

Classical methods often linearize non-linear optimization problems at suitable solutions and use a linear programming technique successively, such as the Frank-Wolfe method or successive linearization methods [10]. Besides, linearization techniques, non-linear problems can be approximated by quadratic or higher-order polynomial functions. Powell's quadratic approximation technique is a popular single-variable search technique [3] in finding an optimum of a non-linear problem.

Another way to approximate a function is to use a set of basis functions and find a weighted sum of such basis functions (finite or infinite numbers of them) as an approximation. Fourier approximation and Walsh function approximations are two such examples. Once such an approximation is known, the individual properties of the optimum of the basis functions may be analyzed to make a guess of the optimum of the approximating function.

The optimization problem can be evaluated exactly at a few finite number of pre-specified solutions in the entire range of the decision variables. Thereafter, an approximating function can be fitted through these function values using regression or some other sophisticated techniques such as artificial neural networks. It is clear that if a large number of solutions are chosen, the approximating function will take a shape similar to the original function. On the other hand, if only a few solutions are chosen, the approximated function will ignore the local details and represent a coarse trend in variation of the function values. If this approximat-

ing function is optimized, it is likely that a GA will proceed in the right direction. However, as a GA tends to converge to the optimum of the approximating function, the approximating function needs to be modified to make a better approximated function from before. Since the population diversity will be reduced while approximating the first approximated function, the second approximating function need *not* be defined over the whole range of the decision variables, as shown in Figure 1. Since the approximating function will

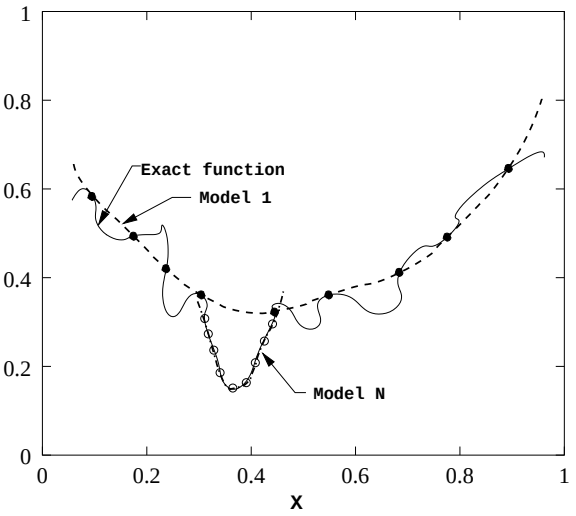


Figure 1: Progressive approximate modeling.

be defined over a smaller search region, more local details can appear in successive approximations. This process may continue till no further approximation results in an improvement in the function value. Although the successive approximation technique described above seems a reasonable plan, care must be taken to ensure that adequate diversity is left while switching from one approximating function to a better one.

Figure 2 outlines a schematic of a plausible plan for the proposed procedure. The combined procedure begins with a

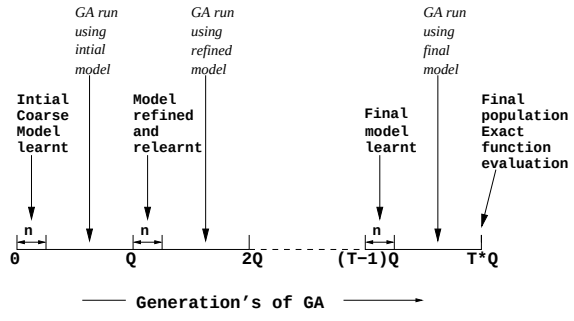


Figure 2: A line diagram of the proposed technique.

set of randomly created $N$ solutions, where $N$ is the population size. Since an adequate size of solutions are required to arrive at an approximated problem, we execute a GA with exact function evaluations for $n$ generations, thereby collecting a total of $N' = nN$ solutions for approximation. At the end of $n$ generations, the approximation technique is invoked with $N'$ solutions and the first approximated problem is created. The GA is then performed for the next $(Q - n)$ generations with this approximated problem. Thereafter, the GA is performed with the exact function for the next $n$ generations and a new approximated problem is created. This procedure is continued till the termination criterion is met. Thus, this procedure requires a fraction $n/Q$ of total evaluations in evaluating the problem exactly. With generations, the approximations continue to happen in smaller regions and, therefore, the training set $N'$ can be reduced in size. We follow a linear reduction in this paper.

If a problem cannot be evaluated exactly, instead some approximations (such as involving FFT or finite element techniques) are needed to evaluate, the parameter $n$ is set to zero and GAs are run for $Q$ generations with the most coarse model (in the case of a finite element method only a few elements can be chosen to start with). It is interesting to note that a set of basis functions with varying importance to local variations can be used as approximating functions here. In such cases, a GA may be started with an approximating function involving only a few basis functions, thereby allowing a quicker computation of the optimal solution. With generations, more and more basis functions can be added to make the model more similar to the actual optimization problem. In this study we concentrate on solving problems for which an exact evaluation method exists but is computationally expensive. However, similar methodology can also be used to solve problem for which no exact evaluation method exists.

### 3.1 Approximation Through Artificial Neural Networks

We propose combining a GA with the artificial neural networks (ANN) as the basic approximating technique for fitness computation. The primary reason for using ANN as the basic approximating tool is its proven capabilities as function approximation tool from a given data set. The multilayer perceptron trained with the back-propagation algorithm may be viewed as a practical vehicle for performing a non-linear input output mapping of general nature [8]. The overall GA-ANN procedure is shown in a flowchart in Figure 3.

An advantage of the proposed technique is its adaptability. Initially the GA population will be randomly spread in the entire search space for the problem undertaken. Since the same information is available in the ANN training database the approximated model generated using this database, will also be very general in nature and hence may miss some finer details about the search space. However as the generations proceed, the GA population will start drifting and focusing on the important regions which it identifies based on the current model. So when the proposed technique updates its model using exact function evaluation for the current generation, it will have more information about the local regions of interest as more population

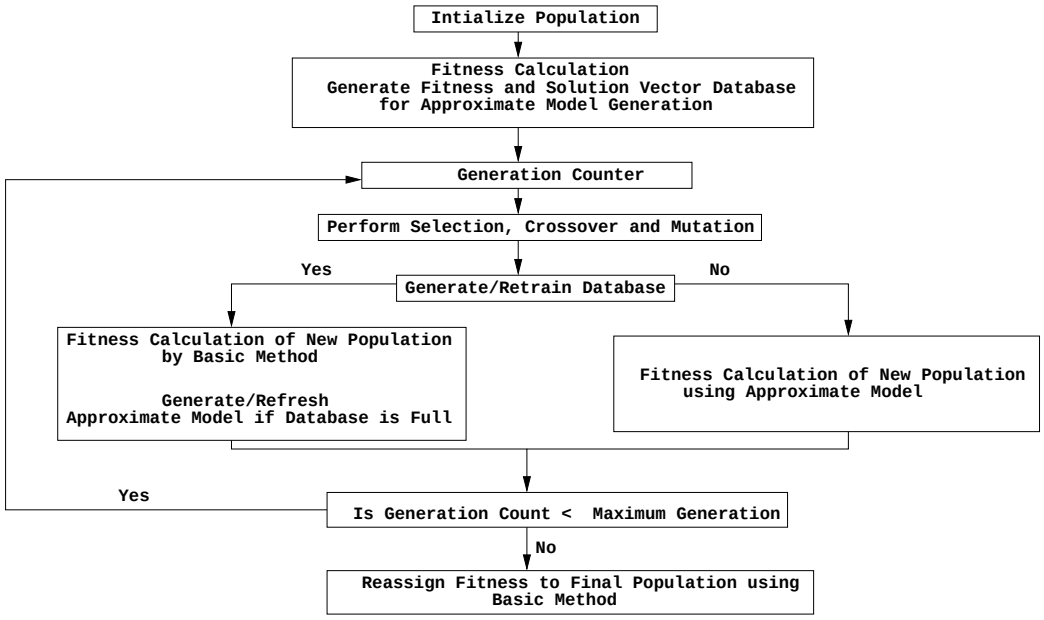


Figure 3: A flow chart of the proposed GA-ANN technique.

members will now be available in those regions than earlier. The ANN will now retrain and update its weights making it learn and adapt to the new smaller regions in the search space. Hence it will give finer refined approximated model to direct the search of GA in the subsequent generations. Thus the proposed technique will adaptively refine the approximated model from coarse to fine approximated model of the optimization problem. It is assumed in this study that the computational time for each function evaluation is so large that the time taken for ANN training is comparatively small. It is also worth mentioning that the proposed technique is equally applicable to single and multi-objective optimization as an additional objective in the optimization problem is equivalent to addition of one more neuron in the output layer of ANN.

## 4 Proof of Principle Results

The proposed technique is tested on a B-spline curve fitting problem here. A saw-tooth function with two teeth is taken as the basic curve to be fitted using B-splines. B-splines enable a better local control of the curve as opposed to the global control achievable in Bezier curves by using a special set of blending functions that provide local influence [14]. They also provide the ability to add control points without increasing the degree of the curve. A sudden change in function values (Figure 4) sets as a challenging task for curve fitting algorithms.

A B-spline with the parameter $k = 3$ produces a polynomial curve of degree two, with $C^1$ continuity for all curve segments and guarantees to pass through the starting and end control points and make tangents at the corresponding

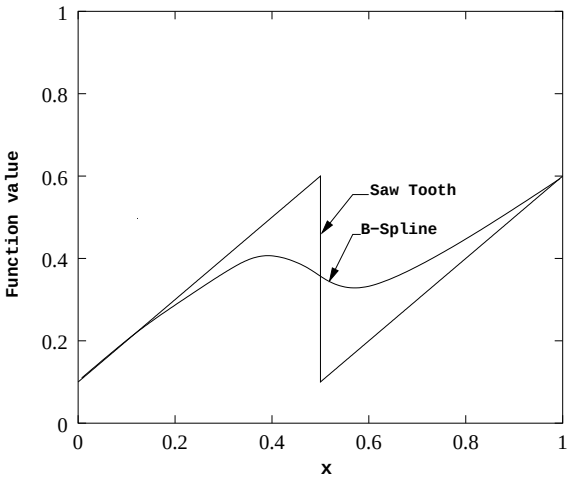


Figure 4: The B-spline curve fitting problem.

line segments. If we edit any control point in the B-spline, it will at-most affect $k$ segments in its vicinity and hence keeping the perturbation local in nature. In the current problem, the number of control points are taken to be 41, thus dividing total $x$ range in 40 equal divisions. However, it is important to note that in order to create meaningful solutions, the first and the last control points for the B-spline is fixed at tooth root height and tooth peak height, respectively, thereby leaving only 39 control points to be treated as decision variables. For the curve fitting problem, the following two conflicting objectives are considered:

- Minimize the error between the saw-tooth curve and

the B-spline fitted curve, and

- Minimize the maximum curvature of the B-spline fitted curve.

Figure 4 also shows a typical B-spline fitted curve to the saw-tooth function. The first objective is calculated as the overall area between the two curves. When this objective is minimized (that is the B-spline fitted curve is very similar to the saw-tooth function), there is bound to be a large curvature near $x = 0.5$. Since in some applications, a large curvature is to be avoided, the sole minimization of the second objective will produce a curve which would be a straight line joining the first ($x = 0$) and the last ($x = 1$) points. As can be seen from the figure that such a curve will make the error (the first objective) large. Thus, the above two objectives constitute a pair of conflicting objectives in the curve fitting problem. Since the problem is posed as a multi-objective problem, we have chosen to use NSGA-II (non-dominated sorting GA - II) [5] to solve the problem.

For any given B-spline curve $\mathcal{S}$, the exact evaluation of the first objective can be achieved in the following manner:

$$F_1(\mathcal{S}) = \int_{x=0}^{x=1} |f_{\text{saw-tooth}} - \mathcal{S}| dx. \qquad (1)$$

Since such a computation is difficult to achieve exactly (mainly because of the non-differential modulus function used in the operand), we compute the above integral numerically by using the Trapezoidal rule. The accuracy of the integral will depend on the number of divisions considered during the integration. The more divisions, the better is the approximation. We have used 400 divisions in the entire range of $x$ and call the procedure an exact evaluation procedure. The second objective can be written as follows:

$$F_2(\mathcal{S}) = \max_{x=0}^{x=1} \frac{\frac{d^2\mathcal{S}}{dx^2}}{\left[1 + \left(\frac{d\mathcal{S}}{dx}\right)^2\right]^{3/2}}. \qquad (2)$$

Since the B-spline curve $\mathcal{S}$ is defined piece-wise, the term for the curvature can be derived exactly for each segment. The term can then be optimized exactly using the first and second-order optimization criteria and the following location of the optimum is found in each B-spline segment:

$$u^* = \begin{cases} 0, & \text{if } u_c \le 0, \\ 1, & \text{if } u_c \ge 1, \\ u_c, & \text{otherwise,} \end{cases} \qquad (3)$$

where the parameter $u_c$ is calculated as follows:

$$\begin{aligned} u_c &= \frac{x_{uu}(x_0 - x_1) + y_{uu}(y_0 - y_1)}{x_{uu}^2 + y_{uu}^2}, \\ x_{uu} &= x_0 - 2x_1 + x_2, \\ y_{uu} &= y_0 - 2y_1 + y_2. \end{aligned}$$

Here $(x_0, y_0)$, $(x_1, y_1)$ and $(x_2, y_2)$ are three control points of each segment. Once the optimal $u^*$ is calculated, the corresponding curvature can be calculated as follows:

$$R = \frac{x_u y_{uu} - x_{uu} y_u}{(x_u^2 + y_u^2)^{3/2}}, \qquad (4)$$

where the first derivatives $x_u$ and $y_u$ are calculated as follows:

$$\begin{aligned} x_u &= (u^* - 1)x_0 + (1 - 2u^*)x_1 + u^* x_2, \\ y_u &= (u^* - 1)y_0 + (1 - 2u^*)y_1 + u^* y_2. \end{aligned}$$

Such computations can be performed for all segments and the maximum curvature of the entire B-spline curve can be determined. For a large number of B-spline segments, many such computations are required, thereby involving a large computation time to evaluate the second objective. If such computations are extended to 3-D curve or surface fitting, the computations become even more expensive.

The ANN module which is used in conjunction with NSGA-II uses two different types of training procedures, namely *batch training* and *incremental training* after every $Q$ generations for next $n$ generations. Thus, the test problem is solved with two different models, namely the incremental ($I$-$Q$-$n$) model and the batch ($B$-$Q$-$n$) model. Each model is tried with various combinations of parameter settings. A three-layer ANN with one hidden layer is used. Input layer has 40 neurons and the output layer has 2 neurons. A momentum factor of 0.1 is used. A unipolar sigmoidal activation function with logistics equal to 0.5 is chosen. In all cases, we have used a permissible normalized RMS error of 0.005. All input data are scaled in [0.1,0.9] and the output data are scaled between zero and one. For initial and final 25% generations, we have used $200n$ and $50n$ training cases, respectively. For intermediate generations, we have linearly reduced the number of training cases. NSGA-II with a population size of 200 and SBX crossover probability of 0.9 with distribution index of 10 and the polynomial mutation probability of 1/39 with a distribution index of 50 are used.

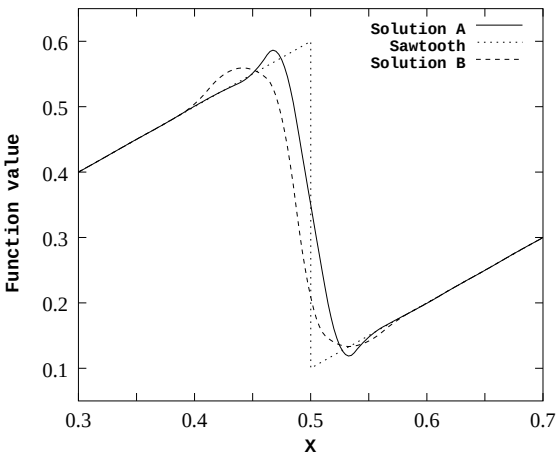


Figure 9: Two extreme non-dominated solutions from the $B$-10-3 model.

In order to investigate the suitable working ranges for the proposed approach, we have tried using various parameter settings, by particularly varying number of hidden neurons

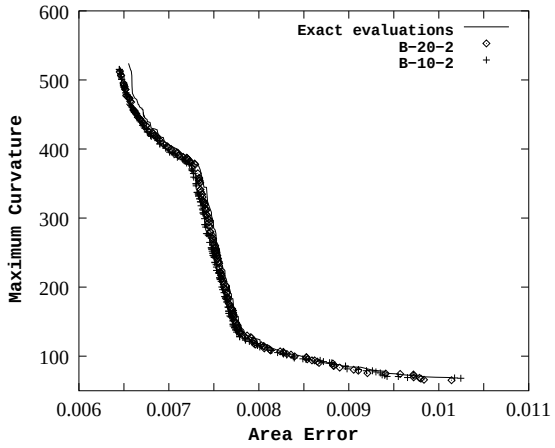


Figure 5: Batch model results trained with 400 patterns.

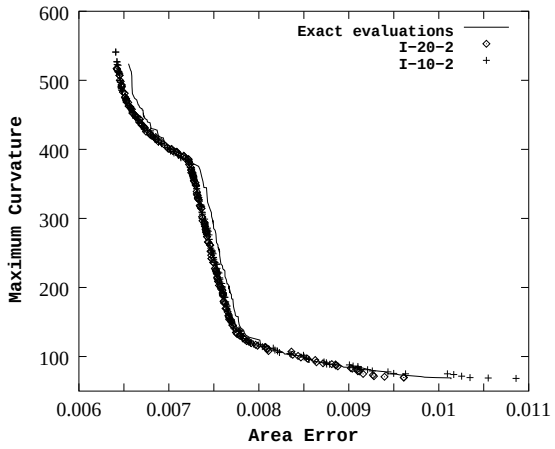


Figure 6: Incremental model results trained with 400 patterns.

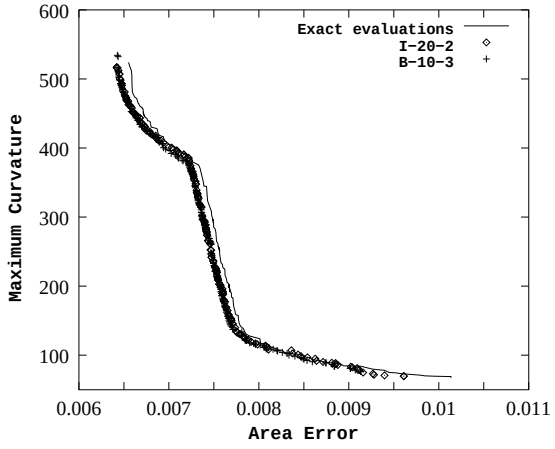


Figure 7: Best of incremental and batch model results.

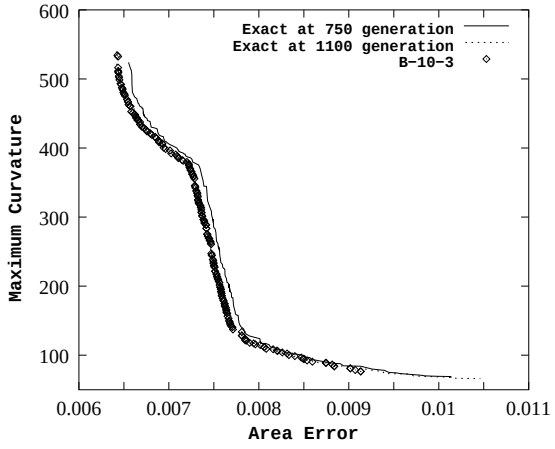


Figure 8: Comparison of the best of the proposed technique and the exact solution at generation 1100.

in the ANN, ANN learning rate, $Q$ and $n$. As a benchmark result, we have run NSGA-II with a population size of 200 using the exact function evaluations (as described above) for 750 generations. The obtained non-dominated front is shown in Figures 5 to 8 using a solid line. The overall function evaluations required in this exact simulation run are $200 \times 750$ or $1,50,000$. All NSGA-II-ANN simulations are also performed for the same number of exact function evaluations.

After the non-dominated solutions are found, they are evaluated using the exact model and plotted in all figures (5 to 8). In all simulations with batch and incremental learning models with different parameter settings, the obtained non-dominated front is better than that obtained using the exact model. The best result for incremental training is found with the $I$-20-2 model, while in the case of batch training slightly better results are found with the $B$-10-3 model. This demonstrates that although approximate models are used, the combined NSGA-II-ANN procedure proposed in this paper is able to find a better non-dominated front than the exact model. In order to investigate how many generations it would take by the NSGA-II with exact evaluations to obtain a front similar to that obtained using the proposed approach, we have continued the NSGA-II run with exact evaluations. Figure 8 shows that the $B$-10-3 model reaches a similar front in about 1,100 generations (except that for larger error values the obtained front is still inferior than that obtained using the proposed approach). In comparison to these evaluations, the approximate model makes a saving of around 32% of exact evaluations.

Figure 9 show two extreme non-dominated solutions obtained by the $B$-10-3 model. The saw-tooth function is also shown in dots. The figure shows that one solution (marked as A) is a better fit with the saw-tooth function, whereas the

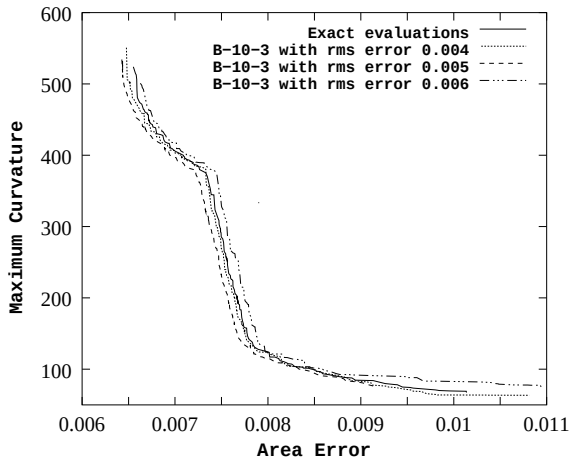


Figure 10: Effect of permissible rms normalized error on B-10-3 model performance.

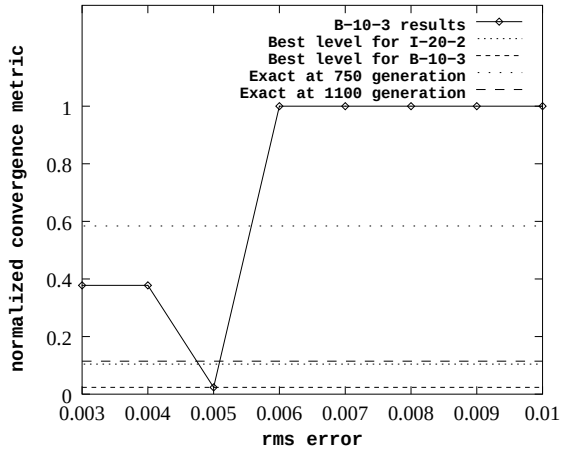


Figure 11: Normalized convergence metric for B-10-3 model.

other solution (marked as B) not a good fit in the vicinity of $x = 0.5$, but produces a smaller curvature. For clarity in showing the nature of the fitted curves near $x = 0.5$, the solutions are plotted in $x \in [0.3, 0.7]$.

### 4.1 Effect of Permissible Normalized RMS Error

The issue of choosing proper permissible normalized rms error for ANN also plays an important role in the proposed NSGA-II-ANN procedure. A too high value of permissible normalized rms error may not adequately approximate the true problem as it will permit too much difference between ANN output and training data. This is referred as poor approximation of the true problem by NSGA-II-ANN procedure. Similarly, a too low value of permissible normalized rms error will lead to over-approximation of the true problem. It is the case in which though the ANN output will match almost exactly with the training pattern data, but will loose its generalization capability. This indicates that there should be a critical value for permissible normalized rms error at which the proposed NSGA-II-ANN procedure should give best performance.

In order to investigate the effect of permissible normalized rms error, various simulations were performed. Figure 10 show one such study. Various values of permissible normalized rms error were tried with $B$-10-3 model in $[0.001, 0.010]$. It was found that permissible normalized rms error less then 0.003 leads to over-approximation of the true problem. Hence NSGA-II-ANN procedure fails to converge for the current problem. Permissible normalized rms error for values upto 0.004 found non-dominated front which was better than exact function evaluations for 750 generation, indicating savings of exact evaluations. At permissible normalized rms error value of 0.005, the performance of proposed NSGA-II-ANN procedure is best as non-dominated front is pushed to the extreme left in the objective space for minimization of both objectives. However, further increasing permissible normalized rms error value at and above 0.006 shows the case of poor approximation of exact problem with no savings in the exact evaluations. For clarity of observation, Figure 10 shows only three such simulations along with exact evaluations for 750 generations. It clearly shows the best performance of B-10-3 model at 0.005 permissible normalized rms error value.

Table 1: Convergence metric calculation.

| Model name | RMS error | Conv. metric | Normalized Conv. metric |
|---|---|---|---|
| $B$-10-3 | 0.003 | 0.012525 | 0.3779 |
| $B$-10-3 | 0.004 | 0.012525 | 0.3779 |
| $B$-10-3 | 0.005 | 0.000779 | 0.0235 |
| $B$-10-3 | 0.006 | 0.033142 | 1.0000 |
| $B$-10-3 | 0.007 | 0.033142 | 1.0000 |
| $B$-10-3 | 0.008 | 0.033142 | 1.0000 |
| $B$-10-3 | 0.009 | 0.033142 | 1.0000 |
| $B$-10-3 | 0.010 | 0.033142 | 1.0000 |
| $I$-20-2 | 0.005 | 0.003458 | 0.1043 |
| Exact-750 | N.A. | 0.019351 | 0.5839 |
| Exact-1100 | N.A. | 0.003795 | 0.1145 |

The visual decision about the performance level of various simulations closely spaced in objective space is extremely difficult. Hence the need to quantify the performance is well recognized [2, 4]. The normalized convergence metric value for various simulations can clearly demonstrate the performance of proposed NSGA-II-ANN procedure and hence can effectively assist in the decision making. The detailed procedure for calculating normalized convergence metric is described elsewhere [4]. As the current problem is a real world problem, the true Pareto-optimal front for which is not known, a non-dominated reference set $P^*$ containing 274 data points is obtained from combined pool of 11 simulations (shown in Table 1) with 2200 data points. For calculating the convergence metric value, first the non-dominated set $F$ of last generation of simulation is identified. Then for each point in $F$, small-

est normalized Euclidean distance to $P^*$ is calculated. Next the convergence metric value is calculated by averaging the normalized distance of all points in the $F$. Lastly, in order to keep the convergence metric within $[0, 1]$, we divide the convergence metric value by the maximum value found among all simulations. Table 1 shows the normalized convergence metric value calculated for various simulations by proposed NSGA-II-ANN procedure. Figure 11 shows the same normalized convergence metric value plot for $B$-10-3 model at various permissible normalized rms error values. The best normalized convergence level obtained with $I$-20-2 model, exact function evaluation for 750 and 1100 generations by NSGA-II are shown by horizontal lines on same plot. It can now be safely concluded that the proposed NSGA-II-ANN procedure with both $I$-20-2 and $B$-10-3 models has outperformed NSGA-II run with exact function evaluations for 1100 generations. Thus a saving of 32% of exact evaluations can be claimed for both $I$-20-2 and $B$-10-3 models. Figure 11 also shows that overall best performance is obtained with $B$-10-3 model.

## 5 Conclusions and Extensions

Many real-world search and optimization problems involve too computationally expensive evaluation procedures to make them useful in practice. Although researchers and practitioners adopt different techniques, such as using a parallel computer or using problem-specific operators, in this paper we have suggested the use of successive approximation models for a faster run-time. Starting from a coarse approximated model of the original problem captured by an artificial neural network (ANN) using a set of initial solutions as a training data set, the proposed genetic algorithm (GA) uses the approximate model. It has been argued that such coarse-to-fine grained approximated models, if coordinated correctly, may direct a GA in the right direction and enable a GA to find a near-optimal or an optimal solution quickly.

The proposed technique is applied to a geometric two-objective curve fitting problem of minimizing the difference between the fitted and the desired curve and of minimizing the maximum curvature in the fitted curve. Simulation results involving a batch learning ANN and an incremental learning ANN obtained using different numbers of training cases and durations of exploiting the approximate model have shown that the proposed GA-ANN approach can find a non-dominated front with about 68% overall function evaluations than that needed if the exact function evaluations were used. Simulations with different values of permissible normalized rms error for ANN have shown that though the proposed approach works successfully for a range of rms error value, there exists a critical value of permissible normalized rms error at which the GA-ANN approach gives the best performance. However, the overall procedure introduces a number of new parameters, the sensitivity of which on the obtained speed-up must be established by performing a more elaborate parametric study.

## Acknowledgments

The first author acknowledges the support from British Telecom under contract number ML832835/CT405106.

## Bibliography

[1] Branke, J., and Schmidt, C.: Faster convergence by means of fi tness estimation. In *Soft Computing Journal*, (in press).

[2] Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms* First Edition, Chichester, UK: Wiley, 2001.

[3] Deb, K.: *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall, 1995.

[4] Deb, K., and Jain, S.: Running performance metrics for evolutionary multi-objective optimization. *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, (Singapore). 2002, pp. 13–20.

[5] Deb, K., Pratap. A, Agarwal, S., and Meyarivan, T.: A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation, 6*(2), 181–197, 2002.

[6] Eby, D., Averill, R. C., Punch III, W. F., and Goodman, E. D.: Evaluation of injection island GA performance on flywheel design optimization. In *Proceedings, Third Conference on Adaptive Computing in Design and Manufacturing.* Springer, 1998.

[7] Goldberg, D. E., Deb, K., and Clark, J. H.: Genetic algorithms, noise, and the sizing of populations. *Complex Systems, 6*, 333–362, 1992.

[8] Haykin, S.: *Neural networks a comprehensive foundation.* second edition, Singapore: Addison Wesley, 2001. pp 208.

[9] Jin, Y.,and Sendhoff, B.: Fitness approximation in evolutionary computation – A survey. In *Proceedings, Genetic and Evolutionary Computation Conference, 2002*. Morgan Kaufmann, 2002, pp 1105-1112.

[10] Reklaitis, G. V., Ravindran, A. and Ragsdell, K. M.: *Engineering Optimization Methods and Applications*. New York: Wiley, 1983.

[11] Sefrioui, M., and Périaux, J.: A hierarchical genetic algorithm using multiple models for optimization. In *Proceedings, 6th International Conference on Parallel Problem Solving from Nature - PPSN VI* . Lecture Notes in Computer Science 1917, Springer 2000.

[12] Poloni, C., Giurgevich, A., Onesti, L., and Prdiroda, V.: Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics. In *Computer Methods in Applied Mechanics and Engineering*, volume 186, 2000, pp-403-420.

[13] Nair, P. B., Keane, A. J., and Shimpi, R. P.: Combining approximating concepts with genetic algorithm-based structural optimization procedures. In *Proceedings,First ISSMO/NASA/AIAA Internet Conference on Approximations and Fast Reanalysis in Engineering Optimization*, 1998.

[14] Zied, I.: *CAD/CAM theory and practice.* New Delhi, India: Tata McGraw-Hill Publishing Company, 2000.