Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

ON THE SCALABILITY OF META-MODELS IN SIMULATION-BASED OPTIMIZATION OF PRODUCTION SYSTEMS

Sunith Bandaru
Amos H.C. Ng

School of Engineering Science,
University of Skövde,
P.O. Box 408, Skövde, SE-541 28, SWEDEN

ABSTRACT

Optimization of production systems often involves numerous simulations of computationally expensive discrete-event models. When derivative-free optimization is sought, one usually resorts to evolutionary and other population-based meta-heuristics. These algorithms typically demand a large number of objective function evaluations, which in turn, drastically increases the computational cost of simulations. To counteract this, meta-models are used to replace expensive simulations with inexpensive approximations. Despite their widespread use, a thorough evaluation of meta-modeling methods has not been carried out yet to the authors' knowledge. In this paper, we analyze 10 different meta-models with respect to their accuracy and training time as a function of the number of training samples and the problem dimension. For our experiments, we choose a standard discrete-event model of an unpaced flow line with scalable number of machines and buffers. The best performing meta-model is then used with an evolutionary algorithm to perform multi-objective optimization of the production model.

1 INTRODUCTION

Discrete-event simulations are used to assess the performance of stochastic process models in production systems. The models involve several process variables which together contribute toward a stochastic response of individual nodes and the system as a whole. The simulation time depends on a number of factors ranging from the parameters themselves, to the complexity of the workflow and the possible number of events at each node. In general, the simulation time increases as more nodes are added to the system. While this may not be a major concern for individual one-off simulations, often the goal is to optimize the process parameters for maximal performance of the system and since iterative optimization requires the evaluation of multiple candidate solutions, the simulation times add-up to give a large computational overhead. Due to the non-linearity and noise associated with simulation models and the lack of gradient information of the responses, most classical optimization methods are either inapplicable or fail to locate the optimum solution reliably, instead converging to a local optimum. Population-based methods, such as genetic algorithms, differential evolution, particle swarm optimization, etc. are more commonly used in such situations owing to their global search characteristics and their ability to converge without the use of function derivatives.

Real-world production system optimization problems involve multiple objectives, like maximization of throughput, minimization of work-in-process, minimization of cost, minimization of inventory, etc. Many of these objectives are conflicting, i.e. any attempt to improve the system with respect to one of the objectives leads to worsening of some other objective. The problem of simultaneously optimizing such

978-1-4673-9743-8/15/$31.00 ©2015 IEEE

objectives is known as *multi-objective optimization* and is mathematically formulated as,

$$\begin{array}{ll} \text{Minimize} & \mathbf{F}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})\} \\ \text{Subject to} & \mathbf{x} \in S \end{array} \tag{1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ are M (> 2) conflicting objectives that have to be simultaneously minimized (maximization of f_i can be handled as minimization of $-f_i$) and the variable vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ belongs to the non-empty feasible region $S \subset \mathbb{R}^n$ formed by the constraints of the problem and the bounds on the variables. Due to the conflicting nature of the objectives, there exist multiple optimal solutions to such problems, each of which is strictly better in at least one of the objectives when compared to all other solutions. This is referred to as the principle of dominance and the solutions are known as Pareto-optimal solutions. The goal of multi-objective optimization methods is to obtain as many near Pareto-optimal solutions as possible (convergence) while also adequately representing the Pareto-optimal front (diversity). Population-based evolutionary algorithms are especially popular in this regard because they can obtain multiple Pareto-optimal solutions in a single algorithmic run.

1.1 Meta-modeling in Evolutionary Computation

From the preceding discussion, it is evident that evolutionary algorithms are popular in the field of simulation-based multi-objective optimization for two main reasons, (i) ability to work without derivatives, and (ii) ability to obtain multiple Pareto-optimal solutions in a single run. Needless to say, they have also been extensively used for production system optimization. However, these methods require a relatively large number of objective function evaluations (simulations), which makes the issue of computational overhead even worse. An alternative is to use functional approximations of the expensive simulation models that can be evaluated fairly quickly for the given candidate solutions. These approximation models are known as meta-models or surrogate models, or simply, surrogates.

Many recent meta-modeling methods have been developed specifically to be used with evolutionary algorithms. Since evolutionary algorithms fall in the category of comparison-based optimizers, the meta-models need to only be good enough for establishing the correct fitness ranking among the solutions, rather than accurately represent the objective function (Runarsson 2004). Jin (2003) provides a comprehensive survey of the available methods in and also highlights key implementation issues. His updated review (Jin 2011) discusses recent advances in using multiple surrogates and the use of surrogates for dynamic, robust and constrained optimization. It also emphasizes the fact that most work in surrogate-assisted optimization is experimental and that no rigorous comparative studies have been reported on the topic. In this paper, we attempt to address this in the context of a scalable production simulation model. To make the comparison as thorough as possible, we consider 10 different meta-modeling methods and assess their performance with respect to function approximation error, solution ranking error and training time.

In the following section, we very briefly describe the 10 meta-modeling methods used in this study. The experimental methodology for evaluating the static meta-models and the scalable discrete-event model used in this study are described in Section 3. Section 4 presents the performance of the meta-modeling methods in terms of accuracy and training time with respect to the number of training samples and size of the simulation model. The primary purpose is to study the scalability of the meta-models on a standard discrete-event model that can be found in most text books related to manufacturing systems. Based on the above results, the best static meta-modeling method is selected and used with a multi-objective optimization algorithm in Section 4.1. Due to space limitations, results with other meta-models are deferred to a future study.

2 META-MODELS

A number of meta-modeling methods can be found in the literature (Jin et al. 2001, Li et al. 2010). Most popular among them are, response surface methods, artificial neural networks, multivariate adaptive regression splines, kriging, radial basis functions and support vector regression. However, there are a few other methods that have not been previously evaluated as meta-models, to the best of our knowledge. Here, we list 10 meta-models along with a short description of their parameters and the settings used in this paper. Most parameters are set to their default values. We use n to represent the number of variables and n_s to represent the number of training samples throughout the paper.

2.1 Multivariate Adaptive Regression Splines (MARS)

MARS is a non-parametric regression technique proposed by Friedman (1991) that can automatically model non-linearities and multi-variable interactions in the training data. The method is non-parametric in the sense that it does not assume a pre-defined form for the fitting function. Instead, the model is derived from the data along with the estimates for coefficients. We use a Matlab implementation of MARS known as ARESLab (Jekabsons 2009), short for Adaptive Regression Splines toolbox for Matlab/Octave. The most important parameters for ARESLab are as follows:

1. *maxFuncs*: Maximum number of basis functions to be included in the model in the forward phase. We use $maxFuncs = 21$ in this paper.
2. *c*: GCV cost per basis function. It acts as a smoothing parameter. Simulation studies suggest values in the range $[2,4]$, with $c = 3$ being "fairly effective" (Friedman 1991). Larger values will lead to fewer knots being placed (i.e. final models will be simpler). We use $c = 3$
3. *cubic*: When a model with continuous first derivatives is desired, the piecewise linear basis functions retained after the backward phase are replaced with piecewise cubic splines. The parameter can either be `true` or `false`. We set *cubic* = `true`.
4. *selfInteractions*: Maximum degree of self interactions for any input variable. We set it to 1.
5. *maxInteractions*: Maximum degree of interactions between input variables. We set it to 2.
6. *threshold*: Another stopping criterion for the forward phase. Larger values of the threshold generate simpler models. We use $threshold = 10^{-4}$ in this paper.

2.2 Kriging or Gaussian Process Regression (DACE)

Kriging models can be thought of as a combination of a global deterministic model and a local stochastic model. The former approximates the target function $f(\mathbf{x})$, similar to a regression method, while the latter allows the kriging model to interpolate the n_s training samples by introducing local deviations. Mathematically, the kriging predictor can be written as,

$$\hat{f}(\mathbf{x}) = \mathscr{F}(\boldsymbol{\beta}, \mathbf{x}) + Z(\mathbf{x}), \tag{2}$$

where $\mathscr{F}(\boldsymbol{\beta}, \mathbf{x})$ can either be a constant term (ordinary kriging), or be equal to zero (simple kriging), or more generally be a linear combination of several polynomial basis functions (universal kriging).

The kriging methodology has been implemented in Matlab as the DACE (Design and Analysis of Computer Experiments) toolbox (Lophaven et al. 2002). The parameters to be set by the user are:

1. *regr*: Three regression models have been implemented, namely `regpoly0`, `regpoly1` and `regpoly2` representing zero, first and second order polynomial choices for $\mathscr{F}(\boldsymbol{\beta}, \mathbf{x})$. We use `regpoly0` when $n_s < n$, `regpoly1` when $n \leq n_s < (n+1)(n+2)/2$ and `regpoly2` otherwise.
2. *corr*: Correlation model. Six correlation models have been implemented, namely `correxp`, `correxpg`, `corrgauss`, `corrlin`, `corrspherical` and `corrspline` representing exponential, generalized exponential, Gaussian, linear, spherical and cubic spline. The popular Gaussian correlation model is used in this paper, $R_{corrgauss}(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \Pi_{k=1}^{n} e^{-\theta_k (x_k^{(i)} - x_k^{(j)})^2}$.
3. $\boldsymbol{\theta}_0$: Initial value for $\boldsymbol{\theta}$. We use $\boldsymbol{\theta}_0 = [2\ 2\ \ldots\ 2]_n$.
4. $\boldsymbol{\theta}_L, \boldsymbol{\theta}_U$: Lower and upper bounds for $\boldsymbol{\theta}$. We use $[0.1\ 0.1\ \ldots\ 0.1]_n$ and $[20\ 20\ \ldots\ 20]_n$ respectively.

2.3 k Nearest Neighbors Regression (kNN)

The k nearest neighbors regression method is one of simplest meta-models to implement. Since there is no model training involved, it is usually among the fastest supervised algorithms. Let $\mathcal{N}_k(\mathbf{x}^{(u)})$ be the set of k nearest neighbors of an *unsampled* point $\mathbf{x}^{(u)}$. The predicted response is usually the mean of the response values at the points in $\mathcal{N}_k(\mathbf{x}^{(u)})$. In this paper an inverse distance weighted response is used with $k = 10$. It is given by,

$$\hat{f}(\mathbf{x}^{(u)}) = \frac{\sum_{i\in\mathcal{N}_k(\mathbf{x}^{(u)})} w_i f(\mathbf{x}^{(i)})}{\sum_{i\in\mathcal{N}_k(\mathbf{x}^{(u)})} w_i}, \quad \text{where} \quad w_i = \frac{1}{||\mathbf{x}^{(u)} - \mathbf{x}^{(i)}||}. \tag{3}$$

2.4 Artificial Neural Networks (NN)

A feedforward neural network consists of input, hidden and the output layers with nodes. The number of nodes in the input and the output layers correspond to the number of variables and responses, while the number of nodes in the hidden layers are up to the user to choose. Each node in a layer is connected to all nodes of the next layer with a certain weight. All nodes (except the input nodes) process their input h (a weighted sum of outputs from the previous layer) using an *activation function* to generate an output v. A neural network is trained by first passing all training instances through the network and calculating the mean squared error. This error is *backpropagated* through the network to update the weights of the node connections. We use Matlab's `fitnet` function with the sigmoidal activation function, $v = (1+e^{-h})^{-1}$, and Levenberg-Marquardt optimization for obtaining the weights. A single hidden layer with 10 nodes is used. A validation set of 20% of training samples is used to terminate the training for generalization.

2.5 Radial Basis Function Network (RBN)

Radial basis function networks are a variation of feedforward neural networks which use a single hidden layer with the number of neurons typically equal to the number of training samples. The main difference however, is the choice of the activation function. Radial basis function networks use Gaussian activation functions centered at points $\mathbf{c}_i$.

$$\phi_i(\mathbf{x}) = \exp\left(\frac{||\mathbf{x} - \mathbf{c}_i||^2}{2\sigma^2}\right) \text{ with spread } \sigma = \frac{\sum_{i,j=1}^{n_s} ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||}{n_s^2}.$$

When $\mathbf{c}_i$ correspond to the n_s training samples, the problem of finding optimal weights reduces to the inversion of a non-singular matrix. We use Matlab's `newrbe` function in this paper.

2.6 Polynomial Regression or Response Surface Methodology (RSM)

Response surfaces are probably the oldest, yet one of the most common meta-modeling methods. They fall under the category of polynomial regression, but mostly use a quadratic model as shown below:

$$\hat{f}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \beta_{ij} x_i x_j + \sum_{i=1}^{n} \beta_{ii} x_i^2 \quad \text{or} \quad \mathbf{Y} = \mathbf{X}\boldsymbol{\beta}. \tag{4}$$

The coefficients are estimated using the ordinary least squares method. First, a *model matrix* or *design matrix* $\mathbf{X}$ of size $n_s \times p$ is formed using the regression terms of the model. For a quadratic model $p = (n+1)(n+2)/2$. The ordinary least squares estimation method cannot uniquely estimate the coefficients when $n_s < p$. Since most experiments in this paper belong to this category, we do not use the RSM meta-model.

2.7 Regularized Regression or Elastic Nets (EN)

The problems with ordinary least squares approach described above can be addressed through *regularization*. The idea is to use a regularization or shrinkage parameter $\lambda \geq 0$ to tune model complexity by penalizing complex models. Elastic nets (Zou and Hastie 2005) combine L_1 and L_2 penalty terms through a parameter $\alpha \in [0,1]$. Instead of minimizing the mean squared error, the elastic net regularized regression minimizes

$$||\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}||^2+\lambda\left[\frac{(1-\alpha)}{2}||\boldsymbol{\beta}||_2^2+\alpha||\boldsymbol{\beta}||_1\right], \quad \text{where} \quad ||\boldsymbol{\beta}||_2^2=\sum_{j=1}^{p}\beta_j^2 \text{ and } \quad ||\boldsymbol{\beta}||_1=\sum_{j=1}^{p}|\beta_j|. \tag{5}$$

Here, $\mathbf{X}$ is the model matrix discussed above. With $\lambda = 0$, the above reduces to ordinary least squares estimation. $\alpha = 0$ corresponds to ridge regression and $\alpha = 1$ gives LASSO. We choose $\alpha = 0.95$.

To the best of our knowledge, elastic nets have not been previously used as meta-models in optimization. We use the popular GLMNET Matlab toolbox (Friedman et al. 2010) in this paper. The toolbox uses 'warm starts' to calculate the regularization path for a sequence of λ values in an extremely efficient way so that the user can choose one of them. In this paper, we use λ_{min}, which corresponds to the minimum cross-validation error model. The number of folds ($nfolds$) for cross-validation and the number of lambda values to generate *nlambda* are user-defined parameters. They are set to 10 and 100 respectively. A quadratic model for $\hat{f}$ is used throughout this paper.

2.8 Support Vector Regression (SVR)

Support vector machine (SVM) based binary classification aims to find the decision boundary that maximizes the margin between two classes in the feature space. Training samples that lie on the margin boundaries are called support vectors, and the classification model is governed only by these training samples. A similar strategy can be employed in a regression model where only training samples with deviations above a threshold ε act as the support vectors. This version of SVM based regression is called ε-Support Vector Regression. We use the popular LibSVM (Chang and Lin 2011) implementation of ε-SVR. The required algorithmic parameters are ε, penalty C and the kernel function. In this paper, we set them to 0.1, 1 and RBF kernel with $\sigma = 1/n$ respectively. SVR performs best when all variables are scaled in $[0,1]$.

2.9 Regression Tree (RT)

Regression tree learning starts with a root node containing all training instances. The algorithm then searches over all intermediate values of all the variables for a split that maximally reduces the mean squared error between the predictions and training responses at the current node. The best split is imposed and the process is repeated recursively. We use Matlab's `classregtree` function for growing a regression tree. We set *NVarToSample* =`all`. At any node, the splitting stops if, (i) the number of observations in it is less than *MinParent*, or (ii) factor of reduction in mean squared error is below a threshold $QEtoler$, or (iii) the number of observations in either of the child nodes is less than $MinLeaf$. These three parameters are set to 10, 10^{-6} and 1, respectively. Fully grown regression trees tend to overfit the training data and exhibit poor predictive performance (low bias high variance). Hence, they are often pruned by either penalizing the number of nodes in the cost function or limiting the number of levels. We set *prune* to `true`.

2.10 Bagged Tree Ensemble and Random Forests (RF)

Bagging is a method of reducing the variance of complex low bias models by using multiple bootstrap samples (random samples with replacement) as training sets for multiple classifiers or regressors. Bootstrap samples have the same size as the original training set. In regression, the prediction is obtained by averaging over all the regressors. Random forests provide an improvement over bagged trees by additionally performing *feature bagging*. A random sample of *NVarToSample* $< n$ variables are considered at each split instead of considering all n variables. Like the bagged tree ensemble, random forests also use a user-defined number of *unpruned* regression trees, *ntrees*. Feature bagging further de-correlates the trees. We use Matlab's `TreeBagger` function in this paper with *ntrees* $= 100$, *NVarToSample* $= \lfloor n/3 \rfloor$, $MinLeaf = 5$ and *prune* =`false`.

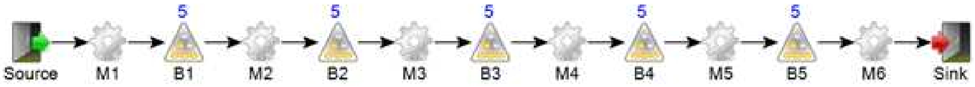

Figure 1: A simple model representing an unpaced flow line with 6 workstations and 5 buffers.

2.11 Boosted Tree Ensemble (BOOST)

While bagging uses multiple independent complex low-bias models and reduces variance, boosting uses multiple nested weak high-variance models and iteratively reduces bias. Each weak model regresses the residuals (difference between the actual function values and the predicted aggregate responses) from previous learners. The process is known as least squares boosting. Typically, the weak models are shallow regression trees. We use Matlab's `fitensemble` function with `LSBoost` option and $ntrees = 100$. The learning rate η is set to the default value of 1.

3 EXPERIMENTAL METHODOLOGY

The simple stochastic simulation model considered in the experiments of this paper represents an unpaced flow line, consisting of s workstations with $s-1$ inter-station buffers (Dallery and Gershwin 1992). For the sake of simplicity, we consider the number of machines in each workstation as one, so that the terms 'workstation' and 'machine' are interchangeable in the following discussions. The productivity of each machine i is governed by its *availability* (α_i), *processing time* (β_i) and *repair time* (γ_i). In the initial state, the workloads of all the workstations are perfectly balanced, each having a processing time of $\beta^{orig} = 80$ seconds per job. All machines have an availability of $\alpha^{orig} = 90\%$ and repair time of $\gamma^{orig} = 300$ seconds. The processing times are assumed to be constant, which is realistic for automated machining processes. The times to failure of the workstations are modeled with exponential distributions and the randomness of the repair times γ_i is modeled using Erlang distributions. As an example, Figure 1 shows the model with $s = 6$ workstations/machines.

In a complex flow line with unbalanced workloads, the detection of bottlenecks is essential for a subsequent improvement of the production rate or throughput. The location of bottlenecks depends on many factors, including the job flow logic, variability and disturbance of the machines and the buffer allocations. Even for a simple, straight flow line with balanced workloads as described above, detecting which workstation(s) to improve in order to increase the overall throughput of the line to a certain level is not a trivial task. The concept of treating this throughput improvement problem as a multi-objective optimization problem of identifying the optimal (minimal) number of steps changes to maximize the throughput was first proposed in (Pehrsson 2013) and later further elaborated in (Ng et al. 2014). In such an optimization formulation, the system throughput (TH) is the primary objective for improvement, so that $f_1(\mathbf{x}) = \max\{TH(\mathbf{x})\}$. The total number of changes, i.e. improvement actions, can be defined as the secondary objective function, $f_2(\mathbf{x})$. We consider three discrete, multi-level improvement variables $\{\alpha_i, \beta_i, \gamma_i\}$ that can each be either set to their original value or to an improved value. The available improvement actions for availability, processing times and repair times and their corresponding step-sizes are:

$$\begin{aligned} \alpha &= \{90, 92, 94, 96, 98\} & \Delta\alpha &= 2 \\ \beta &= \{60, 65, 70, 75, 80\} & \Delta\beta &= 5 \\ \gamma &= \{180, 210, 240, 270, 300\} & \Delta\gamma &= 30 \end{aligned} \tag{6}$$

The second objective, $f_2(\mathbf{x})$, can then be written as a summation of improvements (Ng et al. 2014):

$$f_2(\mathbf{x}) = \min\left\{\sum_{i=1}^{s}\hat{\alpha}_i + \sum_{i=1}^{s}\hat{\beta}_i + \sum_{i=1}^{s}\hat{\gamma}_i\right\}, \text{ where } \hat{\alpha}_i = \frac{\alpha_i - \alpha^{orig}}{\Delta\alpha}, \hat{\beta}_i = \frac{\beta^{orig} - \beta_i}{\Delta\beta}, \text{ and } \hat{\gamma}_i = \frac{\gamma^{orig} - \gamma_i}{\Delta\gamma}.$$

Additionally, in order to simultaneously solve the lean buffer problem (Enginarlar et al. 2005), we optimize the capacity of inter-station buffer spaces, $B_i = \{1, 2, \ldots, 10\}\ \forall\, i \in \{1, \ldots, s-1\}$, by adding a third objective of minimizing the total number of buffers, i.e. $f_3(\mathbf{x}) = \min\{\sum_{i=1}^{s-1} B_i\}$. It is to be noted that the worst objective values for $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ are $12s$ and $10(s-1)$ respectively.

In the experiments considered in this paper, the models developed using FACTS Analyzer 2.0 (Ng et al. 2011) are used to solve the above described three-objective optimization problem. Note that the only objective function that requires simulation is $f_1(\mathbf{x}) = \max\{TH(\mathbf{x})\}$. All meta-models described in the previous section are evaluated with respect to their prediction accuracy and training times on this objective. The number of work stations s is scaled as $s = \{5, 10, 15, 20\}$. The variables in the scalable model are the $3s$ improvement variables $(\alpha_i, \beta_i, \gamma_i)$ and $(s-1)$ buffer variables (B_i). Thus, the problem dimensions scale as $(4s-1)$, i.e. $n = \{19, 39, 59, 79\}$. For each problem dimension, all meta-models are run with different number of training samples, $n_s = \{50, 100, 200, 300, 400, 500\}$ Each of these runs is replicated 10 times by first generating $n_s + n_t$ latin hypercube samples, where n_t of the samples are set aside for testing the trained meta-models. Matlab's `lhsdesign` function is used to generate samples in a unit cube. Each dimension is then normalized to the corresponding variable range and the values rounded to the nearest discrete step as shown in (6).

3.1 Performance Metrics

The number of test samples is set to $n_t = 100$ for all experiments in order to allow a fair comparison in terms of the following metrics:

1. Normalized Root Mean Squared Error (*NRMS*): The throughput response decreases as s increases when all other variables remain unchanged. To be able to compare the deviations of the meta-models across different problem sizes, a normalized RMS value is used which is given by,

$$NRMS = \frac{1}{\sqrt{n_t}} \frac{\sqrt{\sum_{i=1}^{n_t} (f(\mathbf{x}^{(i)}) - \hat{f}(\mathbf{x}^{(i)}))^2}}{(f_{max} - f_{min})}, \text{ where } \begin{array}{l} f_{max} = \max(f(\mathbf{x}^{(1)}), f(\mathbf{x}^{(2)}), \ldots, f(\mathbf{x}^{(n_s+n_t)})) \\ f_{min} = \min(f(\mathbf{x}^{(1)}), f(\mathbf{x}^{(2)}), \ldots, f(\mathbf{x}^{(n_s+n_t)})) \end{array}$$

2. Rank Error (*RE*): As discussed in Section 1.1, evolutionary algorithms only require the ranking among pairs of solutions and not their actual function values. We therefore use the rank error defined in (Joachims 2005) and given by,

$$RE = \frac{SwappedPairs}{{n_t}^2}, \text{ where } SwappedPairs = \left| \{(i,j) : (f(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(j)})) \times (\hat{f}(\mathbf{x}^{(i)}) - \hat{f}(\mathbf{x}^{(j)})) < 0\} \right|.$$

3. Training Time (*TT*): When a dynamic meta-model that is updated frequently is used with EAs, the training time becomes crucial. It is measured in seconds.

4 RESULTS AND DISCUSSION

The best overall meta-model can be obtained through statistical analysis of the three metrics. To improve interpretability and overcome space limitations, we demonstrate the procedure on the normalized root mean squared error (*NRMS*) metric. Table 1 shows the median percentage *NRMS* values of the 10 runs performed for each of the 6 training sample sizes on all 4 problem sizes. The same data can be visualized in Figure 2, which reveals the expected trend that increasing the number of training samples improves the accuracy of the meta-model, while scaling the number of variables ($n = 4s-1$) increases the error for the same number of training samples.

Though the best values are shown in bold in Table 1, the median values are only a representative of the actual distributions of *NRMS*. Thus, a performance ranking among the meta-models cannot be established by simply sorting median values. A more rigorous statistical treatment is required. We adopt the multiple comparison test procedure described by Sachs (2012).

Table 1: Median *NRMS* values from 10 runs for 4×6 experiments on each meta-model.

s	n_s	MARS	DACE	kNN	NN	RT	SVR	RBN	EN	RF	BOOST
5	50	11.573	9.255	9.835	11.863	13.165	9.458	9.114	**8.779**	9.161	10.429
	100	12.807	10.168	13.952	14.477	15.344	12.842	11.971	**9.998**	12.337	11.953
	200	9.539	8.464	11.395	10.248	10.741	10.095	8.494	**7.051**	8.902	8.869
	300	9.211	8.933	11.774	9.870	10.704	9.916	7.860	**6.939**	8.884	8.580
	400	8.852	6.788	10.596	9.240	9.574	8.990	7.495	**6.019**	7.678	7.681
	500	8.321	5.741	9.915	7.898	8.650	8.084	6.367	**5.332**	7.225	7.431
10	50	18.586	20.239	13.518	15.814	18.340	12.915	**12.717**	13.567	13.166	16.646
	100	16.199	**10.977**	12.707	14.918	15.905	12.287	11.271	11.418	12.547	13.789
	200	15.068	**10.196**	13.497	14.791	17.023	11.816	11.568	10.349	12.406	11.732
	300	11.946	9.267	12.223	11.799	15.414	10.285	9.700	**9.103**	11.046	9.463
	400	11.424	8.738	11.627	10.562	15.019	9.196	9.623	**8.474**	10.291	9.393
	500	10.408	8.353	10.449	9.828	13.351	8.415	8.121	**7.770**	9.546	8.330
15	50	17.574	12.023	11.819	21.027	15.411	11.596	12.714	11.752	**11.556**	15.613
	100	15.753	14.099	11.967	16.323	17.116	11.959	**11.196**	12.103	12.041	14.976
	200	14.683	10.272	11.915	14.466	17.287	10.919	10.561	**10.218**	11.767	12.621
	300	14.380	**10.368**	13.123	13.612	17.666	11.310	10.903	10.481	12.475	12.331
	400	13.123	**9.677**	11.938	11.889	16.270	10.394	10.267	9.871	11.186	10.847
	500	12.965	**9.304**	11.584	11.864	15.948	9.917	9.847	9.573	11.038	10.747
20	50	24.927	16.809	16.912	27.942	21.649	16.753	19.173	**16.728**	16.827	22.629
	100	17.428	22.927	14.094	18.932	18.633	**13.083**	13.299	13.471	13.246	17.672
	200	15.972	13.367	14.022	15.992	17.084	13.041	**12.191**	12.752	13.593	15.154
	300	14.188	10.945	12.563	14.219	16.957	11.511	**10.356**	10.926	12.326	13.091
	400	13.943	**10.269**	12.727	13.244	18.071	11.448	11.251	10.566	12.367	11.986
	500	12.175	9.272	11.151	12.180	14.912	9.580	9.332	**9.230**	10.430	9.796

Table 2: Meta-models that are statistically indistinguishable from the best median meta-model in Table 1.

$\{s, n_s\}$	50	100	200	300	400	500
5	DACE, kNN, SVR, RBN, EN, RF, BOOST	DACE, RBN, EN, RF, BOOST	DACE, RBN, EN, RF, BOOST	DACE, RBN, EN, RF, BOOST	DACE, RBN, EN, RF, BOOST	DACE, RBN, EN, RF, BOOST
10	kNN, NN, SVR, RBN, EN, RF, BOOST	DACE, kNN, SVR, RBN, EN, RF, BOOST	DACE, SVR, RBN, EN, RF, BOOST	DACE, SVR, RBN, EN, RF, BOOST	DACE, NN, SVR, RBN, EN, RF, BOOST	DACE, NN, SVR, RBN, EN, BOOST
15	DACE, kNN, SVR, RBN, EN, RF	DACE, kNN, SVR, RBN, EN, RF, BOOST	DACE, kNN, SVR, RBN, EN, RF, BOOST	DACE, SVR, RBN, EN, RF, BOOST	DACE, NN, SVR, RBN, EN, RF, BOOST	DACE, SVR, RBN, EN, BOOST
20	DACE, kNN, RT, SVR, RBN, EN, RF	kNN, SVR, RBN, EN, RF	DACE, kNN, SVR, RBN, EN, RF	DACE, kNN, SVR, RBN, EN, RF	DACE, SVR, RBN, EN, RF, BOOST	DACE, kNN, SVR, RBN, EN, RF, BOOST

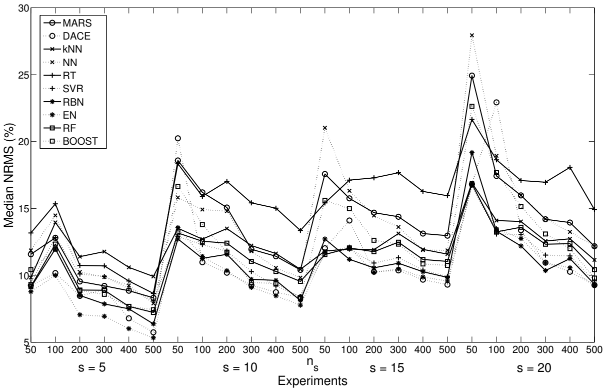

Figure 2: Median *NRMS* for all 10 meta-models across all 24 experiments.

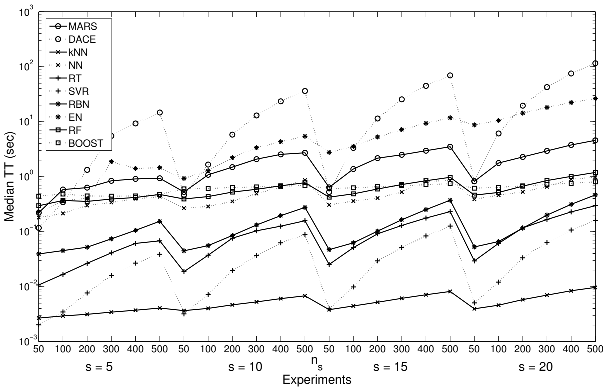

Figure 3: Semi-logarithmic plot of the median TT for all 10 meta-models across all 24 experiments.

The Kruskal-Wallis rank-sum test (also known as 'one-way ANOVA on ranks') is frequently used to test the null hypothesis that groups participating in an experiment come from the same population, i.e. there is no significant difference between them. On applying the Kruskal-Wallis test we find that the medians shown in Table 1 are significantly different in all 24 experiments. Note that a rejection of the null hypothesis of this test only concludes that at least two of the groups are statistically different. In order to find these groups, a post-hoc multiple comparisons test (like Nemenyi test or Dunn test) is performed on the means of rank-sums at $\alpha = 0.05$ level of significance. Table 2 shows a summary of the meta-models that were found to be statistically indistinguishable (according to post-hoc Nemenyi (1963) test) from the meta-model with the best median *NRMS* for all 24 experiments shown in Table 1.

The best overall meta-model can now be obtained by counting the number of times each meta-model appears in Table 2. Figure 4 shows a bar-plot of this count, which we call the *performance score*. A similar bar-plot can be obtained for the *RE* metric as shown in Figure 5. It is clear from these plots that when using meta-models as a direct approximation of the objective functions, RBN and EN are the best choices. However, when the meta-model is to be used as a ranking surrogate in an evolutionary algorithm, RBN, EN, RF and BOOST perform equally well. MARS and RT are the worst performing meta-models in terms of both *NRMS* and *RE*. NN is only slightly better than them with respect to *RE*.

The median training times are best visualized as shown in Figure 3. Note that for all meta-models the training time is most affected by the sample size n_s and less so by the number of variables $n = 4s - 1$. All methods except DACE, SVR and RBN scale approximately linearly with sample size. DACE, SVR and RBN have a computational complexity of $\mathscr{O}(n_s^3)$. The Kruskal-Wallis test followed by post-hoc tests show that the fastest meta-models are kNN, RT, SVR and RBN, all of which have statistically indistinguishable training times across all experiments.

4.1 Meta-model Assisted NSGA-II

Based on the analysis for static meta-models in the previous section, we choose to use EN as the meta-model assisting the popular NSGA-II optimization algorithm for solving the simulation problem described in Section 3. According to Jin (2003), there are two main aspects to the successful use of meta-models for replacing function evaluations in evolutionary methods. Firstly, to ensure that a good global function approximation of the original objective function is achieved, the meta-model should be used together with exact function evaluations. This is known as *model management*. In evolutionary algorithms, model management is achieved via evolution control. Secondly, the quality of the meta-model should be improved as much as possible with the available data. This is achieved through an appropriate *data sampling* procedure. It is important to have an active data sampling method because the region of interest on the fitness landscape changes with the movement of population of the evolutionary algorithm.

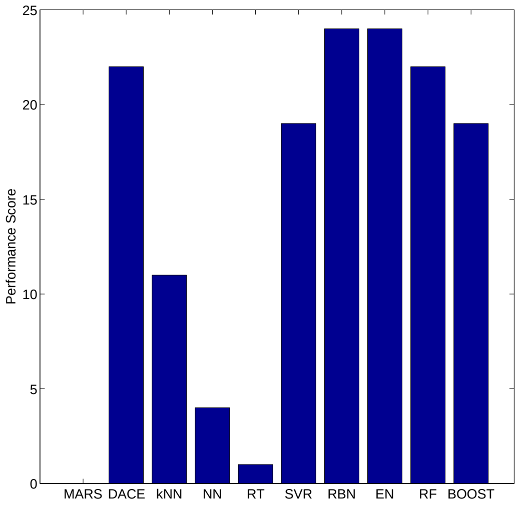

Figure 4: Bar-plot showing the performance score of the meta-models across 24 experiments with respect to *NRMS*.

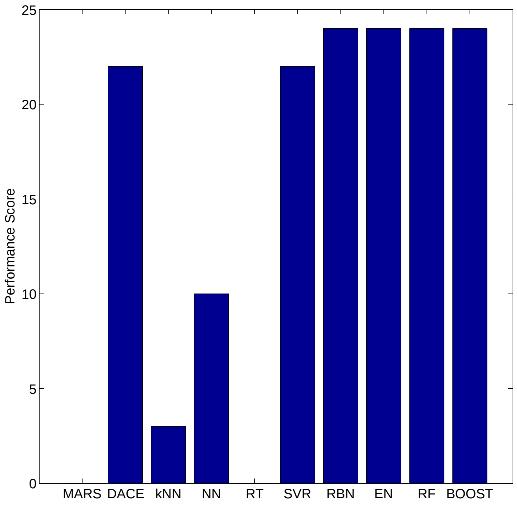

Figure 5: Bar-plot showing the performance score of the meta-models across 24 experiments with respect to *RE*.

NSGA-II (Deb et al. 2002) starts with a randomly initialized population of solutions that undergo selection, crossover and mutation, iteratively over several generations. In each generation, the new population is obtained using non-dominated sorting, which uses the principle of dominance, and crowding distance on the combined parent-child solution set. The meta-model assisted NSGA-II (EN-NSGA-II) framework used in this paper performs evolution control through *controlled generations*, i.e. generations in which all individuals of the population are exactly evaluated. For all other generations, the individuals are evaluated using the meta-model. The controlled generations are evenly spaced and the spacing between them is governed by a user-defined parameter n_{cg}. A fixed evolution control is used, meaning that this parameter remains unchanged during the optimization process. After every controlled generation, the meta-model is updated/recreated using n_s of the available exact function evaluations. This is the most common form of meta-model assisted evolutionary optimization framework.

The parameters associated with meta-model assisted NSGA-II can be classified into three types; those concerning (i) the meta-model, (ii) the meta-modeling framework, and (iii) NSGA-II itself. The parameters of EN are the same as before. The evolution control parameter n_{cg} is set to different values in the set $\{2,5,10\}$. Controlled generations correspond to multiples of n_{cg}. For example, $n_{cg} = 2$ means that every alternate generation is evaluated through exact simulations. The data-sampling parameter is set to $n_s = FE$, which indicates that all exact function evaluations FE available at any given time are used to construct or update the meta-model. The following parameters are used for NSGA-II: (i) Population size $pop = 20$, (ii) Maximum function evaluations $MaxFE = 200$, (iii) Crossover probability $p_c = 0.9$, Mutation probability $p_m = 0.1$, and (iv) Crossover distribution index $\eta_c = 20$, Mutation distribution index $\eta_m = 20$.

The performance of multi-objective optimization algorithms is usually measured using a metric called the hypervolume (Zitzler et al. 2003). It defines the volume of the region bounded by the obtained non-dominated front and a user-defined reference point. The reference point $\mathbf{r}$ is generally chosen to be the worst point or *nadir point* in the objective space. The further away the non-dominated front is from this point, the better is the optimization algorithm that generated the front. A large hypervolume value not only means better convergence to the true Pareto-optimal front, but also a good diversity among the solutions. Thus, algorithms that can consistently generate non-dominated fronts with larger hypervolume values are said to be better performing. Table 3, on the left, shows the median hypervolume values obtained using NSGA-II (without meta-model) for different problem dimensions of the simulation model in Section 3 and the corresponding reference points (r_1, r_2, r_3). On the right, Table 3 shows the median hypervolume values

Table 3: Median hypervolume values from 10 runs of NSGA-II and EN-NSGA-II.

s	r_1	r_2	r_3	NSGA-II	EN-NSGA-II		
					$n_{cg} = 2$	$n_{cg} = 5$	$n_{cg} = 10$
5	30	60	40	23666.267	25781.788	26176.278	**27622.797**
10	26	120	90	84221.428	94979.512	106348.852	**106757.651**
15	25	180	140	186816.956	220789.390	243580.048	**247754.880**
20	24	240	190	314789.374	372899.851	**414676.387**	402450.201

obtained from 10 independent runs of EN-NSGA-II for all 4 problem dimensions. The hypervolume values are calculated for $n_{cg} = \{2,5,10\}$ with the same reference points. The best hypervolume values are shown in bold. It is evident that meta-modeling improves the performance of NSGA-II. However, more interesting is the effect that n_{cg} has on this improvement. By increasing n_{cg}, the user makes a gamble of trusting the meta-model for longer number of generations. This pays-off for $s = 5,10$ and 15, as seen by the increase in hypervolume values with n_{cg}. However, for $s = 20$, it is observed that the hypervolume value drops at $n_{cg} = 10$. In the absence of newly sampled exact function evaluations, the population of EN-NSGA-II is misled by increasingly inaccurate function values from the meta-model, which causes this decrease in the hypervolume. This indicates that higher the dimensionality of the training data, less trustworthy is the elastic net meta-model on samples from a moving population. A similar behavior is expected of the other 9 meta-model assisted NSGA-II algorithms and will be pursued in a future study. An adaptive evolution control can be used to compensate for this effect and will also be explored in the future.

5 CONCLUSIONS

To our knowledge, this paper is the first attempt in studying the scalability of several meta-modeling methods in simulation-based optimization. In addition to popular meta-models like kriging and neural networks, we have also studied some lesser used methods like boosting, and a regularized regression technique called elastic net that has surprisingly never been studied previously as a meta-model to be used with optimization. The present paper bases its conclusions on rigorous statistical analysis. First, latin hypercube samples were generated for different problem sizes of a scalable discrete-event simulation model of an unpaced flow line. The meta-models were assessed with respect to their mean squared errors, ranking errors and training times. These metrics were analyzed across a series of experiments involving different number of variables and sample sizes. The most interesting finding is that elastic nets significantly outperform all other methods except radial basis function networks and in general give the best median performance.

A generic meta-model assisted NSGA-II framework is used to evaluate elastic net during optimization. The interface uses two parameters that define the evolution control and the training samples. It is observed that the elastic net meta-model based NSGA-II (EN-NSGA-II) performs statistically better than NSGA-II even with a very small budget size of 200 evaluations. This shows that elastic net is capable of learning the throughput landscape of typical discrete-event simulation models even when the models are scaled in the number of workstations. The fidelity of the meta-model is found to be dependent on both problem size and the evolution control parameter. The effects of an adaptive evolution control and the data sampling parameter on other variants of meta-model assisted multi-objective algorithms remain to be seen.

REFERENCES

Chang, C.-C., and C.-J. Lin. 2011. “LIBSVM: A Library for Support Vector Machines”. *ACM Transactions on Intelligent Systems and Technology* 2 (3): 1–27.

Dallery, Y., and S. B. Gershwin. 1992. “Manufacturing Flow Line Systems: A Review of Models and Analytical Results”. *Queueing Systems* 12 (1-2): 3–94.

Deb, K., S. Agarwal, A. Pratap, and T. Meyarivan. 2002. “A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II”. *IEEE Transactions on Evolutionary Computation* 6 (2): 182–197.

Enginarlar, E., J. Li, and S. M. Meerkov. 2005. “How Lean Can Lean Buffers Be?”. *IIE Transactions* 37 (4): 333–342.

Friedman, J. H. 1991. “Multivariate Adaptive Regression Splines”. *The Annals of Statistics* 19 (1): 1–67.

Friedman, J. H., T. Hastie, and R. Tibshirani. 2010. “Regularization Paths for Generalized Linear Models via Coordinate Descent”. *Journal of Statistical Software* 33 (1): 1–22.

Jekabsons, G. 2009. “ARESLab: Adaptive Regression Splines Toolbox for MATLAB”. Available from http://www.cs.rtu.lv/jekabsons/.

Jin, R., W. Chen, and T. Simpson. 2001. “Comparative Studies of Metamodelling Techniques Under Multiple Modelling Criteria”. *Structural and Multidisciplinary Optimization* 23 (1): 1–13.

Jin, Y. 2003, October. “A Comprehensive Survey of Fitness Approximation in Evolutionary Computation”. *Soft Computing* 9 (1): 3–12.

Jin, Y. 2011. “Surrogate-Assisted Evolutionary Computation: Recent Advances and Future Challenges”. *Swarm and Evolutionary Computation* 1 (2): 61–70.

Joachims, T. 2005. “A Support Vector Method for Multivariate Performance Measures”. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, 377–384. New York, NY: ACM.

Li, Y., S. Ng, M. Xie, and T. Goh. 2010, September. “A Systematic Comparison of Metamodeling Techniques for Simulation Optimization in Decision Support Systems”. *Applied Soft Computing* 10 (4): 1257–1273.

Lophaven, S. N., H. B. Nielsen, and J. Søndergaard. 2002. “Aspects of the Matlab Toolbox DACE”. Technical report, Technical Univeristy of Denmark, Lyngby, Denmark.

Nemenyi, P. 1963. *Distribution-Free Multiple Comparisons*. Ph. D. thesis, Princeton University.

Ng, A. H., J. Bernedixen, and P. Leif. 2014. “What Does Multi-Objective Optimization Have to do with Bottleneck Improvement of Production Systems?”. In *Proceedings of the 6th International Swedish Production Symposium*.

Ng, A. H., J. Bernedixen, M. U. Moris, and M. Jägstam. 2011. “Factory Flow Design and Analysis Using Internet-Enabled Simulation-based Optimization and Automatic Model Generation”. In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. Creasey, J. Himmelspach, K. White, and M. Fu, 2181–2193. Piscataway, New Jersey: IEEE.

Pehrsson, L. 2013. *Manufacturing Management and Decision Support Using Simulation-based Multi-Objective Optimisation*. Ph. D. thesis, De Montfort University.

Runarsson, T. P. 2004. “Constrained Evolutionary Optimization by Approximate Ranking and Surrogate Models”. In *Parallel Problem Solving from Nature - PPSN VIII*, 401–410. Berlin Heidelberg: Springer.

Sachs, L. 2012. *Applied Statistics: A Handbook of Techniques*. Springer Science & Business Media.

Zitzler, E., L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca. 2003. “Performance Assessment of Multiobjective Optimizers: An Analysis and Review”. *IEEE Transactions on Evolutionary Computation* 7 (2): 117–132.

Zou, H., and T. Hastie. 2005. “Regularization and Variable Selection Via the Elastic-net”. *Journal of the Royal Statistical Society* 67 (2): 301–320.

AUTHOR BIOGRAPHIES

SUNITH BANDARU is currently a postdoctoral researcher at University of Skövde, Sweden. He obtained his Ph.D. in the area of evolutionary optimization and machine learning from Indian Institute of Technology Kanpur, India. His research interests are knowledge discovery, data-mining and machine learning, multi-objective optimization, evolutionary algorithms, simulation-based optimization and meta-modeling. His email address is sunith.bandaru@his.se.

AMOS H.C. NG is a Professor at the University of Skövde, Sweden. He holds a Ph.D. degree in Computing Sciences and Engineering. His main research interest lies in applying multi-objective optimization for production systems design and analysis. His email address is amos.ng@his.se.