Surrogate-Assisted Learning of Neural Networks

Jörg Stork, Martin Zaefferer, Andreas Fischbach, Frederik Rehbach, Thomas Bartz-Beielstein

SPOTSeven Labs, TH Köln
Steinmüllerallee 1, 51643 Gummersbach
E-Mail: firstname.lastname@th-koeln.de

Introduction

Surrogate-assisted optimization has proven to be very successful if applied to industrial problems. The use of a data-driven surrogate model of an objective function during an optimization cycle has many benefits, such as being cheap to evaluate and further providing both information about the objective landscape and the parameter space. In preliminary work, it was researched how surrogate-assisted optimization can help to optimize the structure of a *neural network* (NN) controller [7]. In this work, we will focus on how surrogates can help to improve the direct learning process of a transparent feed-forward neural network controller. As an initial case study we will consider a manageable real-world control task: the *elevator supervisory group problem* (ESGC) using a simplified simulation model [3]. We use this model as a benchmark which should indicate the applicability and performance of surrogate-assisted optimization to this kind of tasks. While the optimization process itself is in this case not considered expensive, the results show that surrogate-assisted optimization is capable of outperforming metaheuristic optimization methods for a low number of evaluations. Further the surrogate can be used for significance analysis of the inputs and weighted connections to further exploit problem information.

Motivation

Recent advancements in robotics and control have shown, that methods from the field of computational intelligence are becoming more and more

significant. Robot control policies are no longer just trained by machine learning algorithms. Rather, robots learn how to solve a certain task by themselves, e.g., by methods of evolutionary robotics [4]. In an real world environment evolutionary learning of control policies can be costly, as the fitness of a certain robot action can only be evaluated after a sequence of time steps, which can easily be in minutes or hours. Thus, these learning processes pose a difficult optimization problem and standard learning methods are not suitable for the time requirements of these tasks. Neural networks are a well established type of controller in evolutionary robotics. Here, the set of coefficients and the topology of the network need to be optimized for optimal performance. More recent and sophisticated approaches for developing and learning of controllers, such as *neuroevolution of augmenting topologies* [18] were invented to handle these optimization processes, but they still need many evaluations to adapt the neural networks.

- Our hypothesis is that assisting this learning process by means of surrogate-assisted optimization, which has proven to be able to perform significantly well in expensive industrial optimization tasks [14, 15], should be beneficial.
- As a second hypothesis, we assume that these surrogate models can help to retrieve additional useful information about the objective function, e.g., importance of certain inputs.

We want to test this hypothesis based on experiments with a small real-world task simulator, which is implemented as a simple neural network and not expensive to evaluate. The results can be transferred to more sophisticated tasks, like a real world learning process. The results should indicate the applicability and basic performance of surrogate-assisted optimization methods in comparison to state-of-the-art optimization algorithms. The variable importance information provided by the surrogate models is also analyzed with regard to their usefulness. For instance, variable importance could be helpful to identify especially important or defective inputs sensors of a physical controller, e.g., in an evolutionary robotics task.

The Elevator Supervisory Group Problem

General Description

Today, elevator systems are present everywhere in urban areas. They need to be optimized to achieve the desired service quality in terms of waiting time for the customers, as well as in terms of energy efficiency. They are controlled by an elevator group controller, which assigns the elevator cars to certain floors and destinations on basis of the customer service calls. The ESGC problem as introduced by [3] is a so-called destination call system, where the customer can choose their desired destination on the floor level outside the elevator cars. In the introduced problem instance, the controller is implemented as a sophisticated *neural network* NN, where the specific structure and weights depict a certain control strategy. The optimization of these weights imposes a set of challenges, which render this task highly complex:

- The topology of the fitness function is to a high extent non-linear as well as multi-modal.
- The traffic load is dynamic and stochastic, as customers do not arrive in a deterministic manner.
- Gradient-based methods cannot be applied successfully to this optimization problem.
- The simulator is computational expensive, which limits the number of function evaluations.

As consequence of the complexity of such simulators [3] introduced a simplified validation model of an ESGC system, the *sequential ring*(S-Ring).

S-Ring Perceptron Simulator

The S-Ring was introduced to benchmark different ESGC algorithms independent of a certain elevator/floor configuration. It uses a simplified NN to control the elevators, where the connection weights can be modified and represent the variables of an optimization problem. Each weight setting

will result in a certain control strategy which is tested on simulations of different traffic situations. The S-Ring has low computational costs, which allows us to use an ESGC instance as a benchmark for a large variety of algorithms. Using different traffic situations will lead to a fitness function which is subject to noise. The S-Ring optimization problem can be defined as follows [3]:

$$F(n, m, p, \vec{w}) = E\left(\sum_{i}^{t} \vec{c}_i\right) \tag{1}$$

where n is the number of elevators, m the number of floors, p the probability of an arriving customer per floor and $\vec{w}$ the NN weight vector, which depicts the control policy. This objective function evaluates the average waiting time of all customers $\vec{c}_i$ during a simulated traffic situation with t steps. For a given set of n, m, p the performance is only influenced by the weight vector of the NN controller. Thus, the simplified problem, as further used during this paper, can be written as $F = F(\vec{w})$. The parameters n, m, p were set as follows: Table 1 also displays the number of time steps for a single

Table 1: S-Ring Configuration

nFloors	nElevators	probNewCustomer	nIterations
6	2	0.3	10000

simulation run, which was set rather high to simulate a longer period. For each simulation run, the exact same period was used, resembling a certain fixed time-frame, e.g. a certain day in a year. By choosing a fixed time frame, we removed the noise of the problem, which renders the problem simpler to optimize. Moreover, the problem was adapted by setting the desired customer service quality of the ground floor to a high priority, while the second floor was set to a lower priority. This should simulate a typical real world hotel scenario, where it is wanted that arriving customers in the lobby are fast served. The second floor displays an internal service area, which is of low priority for the quality of service. As a side effect, this reduces the dimensionality of the optimization problem from 12 to 10.

Methods for Learning of Neural Networks

A standard method for learning NN controller is *backpropagation*. Backpropagation optimizes the weights by utilizing a set of training data that contains input values with corresponding outputs. In case of the S-ring optimization, our task is not machine learning, but to find the (single) global optimum for the given fitness topology of a time dependent simulation problem. We receive a fitness value only after evaluating the weights in a designated simulation run. This means, there is no clear mapping from input to output data, as the output only defines a certain control policy and the final action changes dynamically in every time step. Thus we will need to use more sophisticated methods: metaheuristic optimization and surrogate-assisted optimization.

Metaheuristic Optimization

Metaheuristics are sophisticated heuristics, which are often inspired by nature. They utilize stochastic processes (randomization) and usually do not require any gradient information. Metaheuristics are known to be general solvers which apply to a large variety of *global* problems without needing a priori information. They are suitable for highly non-linear and multi-modal problems, as well as so-called *black-box* problems, where no information about the topology of the objective function is known. No algorithm is able to deliver their best performance for every problem without adapting their control parameters; By parameter tuning [2, 6], we can exploit beneficial parameter settings, but it is very time-demanding. To provide reliable results without putting a lot of effort into algorithm tuning, we selected four different state-of-the-art R implementations of common metaheuristics from the range of *simulated annealing* methods and *evolutionary algorithms* for our comparison. Simulated annealing [9] is inspired by annealing processes in metallurgy, where materials are heated and cooled to change their physical structure. Simulated annealing follows the base principle of an greedy stochastic algorithm, but implements a control strategy which allows to accept also solutions with lesser fitness. This allows to escape local optima and establishes a global search strategy. Evolutionary algorithms [1] are based on the principles of natural selection: in each generation, a population of individuals (e.g. solutions $\vec{w}$) is evolved

by mutation, recombination and selection steps. The selected packages are *DEoptim*, *GA*, *GenSA* and *genoud*. DEoptim and GA were chosen due to personal preference, while the two latter were chosen based on the survey on *Continuous Global Optimization in R* by Mullen [12]. GenSA and genoud performed best on a set of different optimization problems.

- **DEoptim** [13] is an R-implementation of the differential evolution algorithm [19], which belongs to the class of evolutionary algorithms. It is designed for global optimization using real vectors.
- **GA** [16] is a package which implements an genetic algorithm an allows optimization of real and integer problems.
- **GenSA** provides a version of generalized simulated annealing [20].
- **rgenoud** [11, 17] is an R-package which provides and implementation of a so-called *hybrid* algorithm. This algorithm combines evolutionary algorithms with the derivative-based quasi-Newton method *Broyden-Fletcher-Goldfarb-Shanno* (BFGS).

Surrogate-Assisted Optimization

Surrogate-assisted optimization algorithms employ data driven models to lighten the burden of expensive objective function evaluations. One framework for surrogate-assisted optimization is Sequential Parameter Optimization (SPO) [2]. SPO provides a flexible framework that employs methods from the fields of design of experiment, optimization, and statistics. In essence, SPO starts by generating an initial design of experiment, then builds a surrogate model (e.g., a linear model or Kriging). Then, the surrogate model is optimized to suggest a promising candidate solution, which is afterwards evaluated by the expensive objective function. These last steps (model building, optimization and evaluation) are iterated until some budget of evaluations is exhausted. Figure 1 shows the optimization cycle for the underlying ESGC problem.

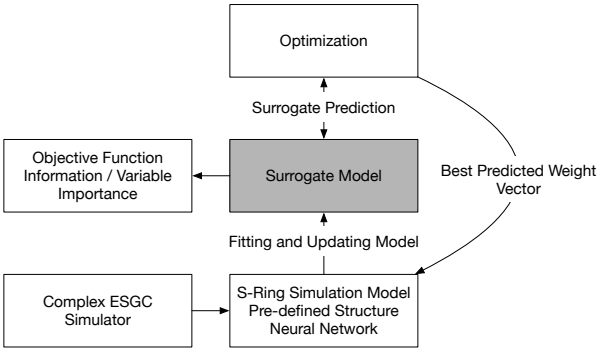

Figure 1: Surrogate-Assisted Optimization Cycle. The ESGC Simulator is approximated by the S-Ring simulator. The fitness topology is fitted by the surrogate on basis of the initial design and sequential updates. The sequential weight vectors are computed by an optimization of the surrogate.

The experiments make use of SPOT, the R implementation of SPO. For this study, we have chosen to investigate three different surrogate models within the SPO framework.

- **Second order model with step-wise regression:** Firstly, we build second order linear regression models. The model is first build with all first order effects, quadratic effects as well as second order interactions. E.g., for two parameters x_1 and x_2 a model of the form $y(x) = c_1x_1 + c_2x_2 + c_3x_1^2 + c_4x_2^2 + c_5x_1x_2$ is determined. This full model is further refined by backwards, stepwise variable selection based on the Akaike information criterion. The stepwise variable selection is skipped whenever the data size is insufficient. While the resulting models are comparatively simple, one advantage is the comparatively low computational effort.
- **Random Forest:** Secondly, we use a Random Forest [5] model. Random Forests are ensembles of decision trees. We use the default settings of the randomForest R-package [10]. Random Forests are able to learn non-linear dependencies in the data, are typically numerically robust and fast to compute, and can handle discrete input variables.

- **Kriging:** Thirdly, a Kriging model (also known as Gaussian process regression) is employed. Kriging assumes that the observed data is the result of a stochastic process. We use an implementation from the R-package SPOT. The implementation is loosely based on Matlab code by Forrester et al. [8]. The correlation of samples is modeled via an exponential correlation function $\text{cor}(x, x') = \exp(-\sum_{i=1}^{n} \theta_i |x_i - x'_i|^{p_i})$. The vectors x and x' are samples, or candidate solutions of the optimization problem. The parameters $\theta_i > 0$ and $1 \leq p_i \leq 2$ are determined by maximum likelihood estimation. Forrester et al. provide a detailed and easy to follow description of Kriging and related methods [8]. Of the three models, Kriging requires the largest computational effort, yet produces the potentially most accurate model.

Variable importance

Most black-box optimizers tend to deliver only the best found parameter settings found within the available budget of objective function evaluations. Hence, these optimizers do no provide any information on what they have learned about the importance of the input variables during the optimization process. An advantage of the surrogate model techniques applied in this study is the provision of some knowledge beyond the best found parameter setting. For example, the estimated model coefficients or the change of the prediction accuracy (i.e. the model error) by permuting variable values may provide strong indicators for the importance of each input variable.

- **Linear Regression Models:** In linear regression models, p-values can be taken to analyze the significance of a variable. However, statistical significance does not automatically mean a large impact of the variable on the result. The values of the regression coefficients can be compared instead. Larger coefficient values account for larger impact of the corresponding variable for the outcome. This has to be used carefully, especially if the scales of variables differ. The data must be standardized first, to enable a safe comparison of the regression coefficients to judge the variables importance.
- **Random Forest:** Variable importance can be estimated by computing the out-of-bag error first. Afterwards a permutation of the values

of one variable is computed randomly. This keeps the distribution of the values equal. In the next step the out-of-bag error is computed again and the result compared to the initial out-of-bag error. If the error (e.g., mean squared error) varies a lot, the variable can be seen as important. With this process, a ranking of the importance of the variables can be computed.

- **Kriging:** The width parameter θ_i determines how far the influence of each sample point spreads in dimension i. In detail, the larger the width parameter is, the faster are the potential changes in the predicted value. The smaller the width parameter is, the slower are the potential changes in the prediction. The descending order of the θ values give an indicator of the variable importance.

Experiments

Our stated hypothesis is tested by performing a benchmark on the S-Ring problem. The performance of a *random search* algorithm will be added as a baseline comparison. All four metaheuristic algorithms and further the surrogate-assisted optimization with three different models are compared on basis of their total objective function evaluations to simulate the performance on a possible expensive function. Thus, the smallest number of evaluations is set to 100, which is a common limit for optimization of expensive objective functions. The maximum number of tested evaluations for the metaheuristics is set to 1_{e+5} to see the convergence behavior on the objective function. The maximum number of surrogate-assisted optimization runs are limited to 1000 total objective function evaluations and 10 percent of this budget are used for the initial *latin hypercube sampling*. A single optimization iteration can, due to the model fitting, model optimization and prediction, become very computationally expensive for large sample sizes. As all tested algorithms are stochastic, each experiment is repeated 20 times. As previously mentioned, for all tested algorithms the parameter settings, beside the iterations and populations size to set an exact number of evaluations, are not changed and use the pre-set default settings. The experimental setup is summarized in table 2.

Table 2: Experimental Setup

Algorithm	No. Evaluations	popSize	maxIter
RandomSearch	$100, 1000, 1_{e+5}$	$100, 1000, 1_{e+5}$	1
Metaheuristics:			
GenSA	$100, 200, 500, 1000, 1_{e+5}$	/	/
DEoptim	$100, 200, 500, 1000, 1_{e+5}$	5,5,10,10,100	9,19,24,49,499
GA	$100, 200, 500, 1000, 1_{e+5}$	10,10,20,50,50	10,20,25,20,2000
genoud	$100, 200, 500, 1000, 1_{e+5}$	10,10,20,50,1000	10,20,25,20,100
Surrogate-Assisted: Model	No. Evaluations	initDesignSize	Optimizer
SecondOrderLM	$100, 200, 500, 1000$	10,20,50,100	DEoptim
RandomForest	$100, 200, 500, 1000$	10,20,50,100	DEoptim
Kriging	$100, 200, 500, 1000$	10,20,50,100	DEoptim

Results and Discussion

Benchmark Comparison

Figure 2 shows the benchmark results for the different combinations of algorithms and number of evaluations:

- **Metaheuristics:** For 100 evaluations, *GA* and *genoud* perform better than random search, while *DEoptim* and *GenSA* show inferior results. The methods significantly improve with a rising number of evaluations and are able to outperform random search. For instance, for 1000 evaluations, most method (except GenSA) perform better than the random search method, which is also the case for 1_{e+5} evaluations. The long run results (1_{e+5}) also indicate that *GenSA* and *DEoptim* seem to converge to a global optimum, while *genoud* and *GA* show significantly inferior results. A large number of evaluations seems to benefit *GenSA* the most, as with less evaluations it is in all cases outperformed by the other algorithms. This can be strongly connected to the chosen default parameter settings.

- **Surrogate-Assisted Optimization:** For 100 evaluations, the surrogate-assisted optimization outperforms the metaheuristics significantly. Particular *Kriging* is performing on a level equivalent to this of 1000 random search evaluations and near 500 metaheuristic evaluations. This picture becomes even clearer for 200 evaluations, where *Kriging* again improves and surpasses random search with 1000 evaluations. For 1000 evaluations, *Kriging* reaches the level of 1_{e+5} random search evaluations and thus outperforms all metaheuristic algorithms on this level. The *second-order linear model* also performs well, but is not able to show considerable improvements in comparison to the metaheuristics.*Random forest* shows poor results for larger evaluations sizes, where it is inferior to random search.

The good results of the *Kriging* surrogates can be explained by their strong ability of fitting non-linear landscapes and their general good interpolation ability. While the *second-order linear model* can fit non-linear behavior so a certain extend, they are not fit to build global surrogates of highly multi-modal landscapes. The poor performance of *random forest* is due to the pure continuous nature of the problem and the bad interpolation abilities of these models.

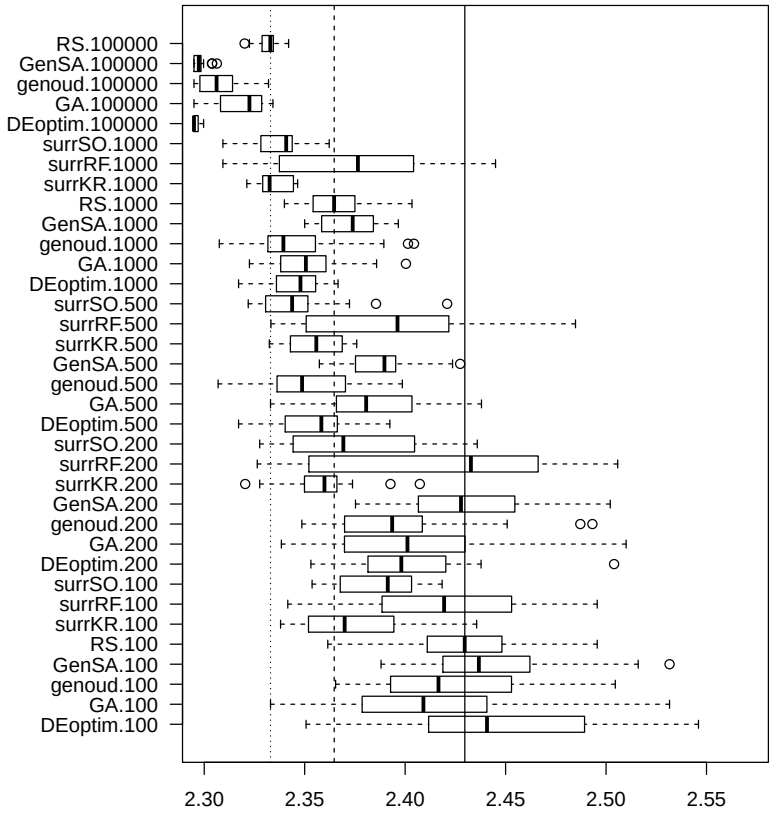

Figure 2: S-Ring Simulator Benchmark Results. The algorithms with their respective number of evaluations are shown on the y-axis, the x-axis shows the achieved fitness. SurrSO uses the second order model, SurrRF the random forest model and SurrKR the Kriging model. The vertical lines represent the baseline, where the solid line is the median random search fitness for 100, the dashed line for 1000 and the pointed line for 100000 evaluations

Model Variable Importance

To check the usefulness of the variable importance, we used three of the best models, which were fitted with 1000 evaluations.

- The *second-order linear model* has a multiple R-squared of 0.869 and an adjusted R-squared 0.8611, thus it is able to explain a high extent of the underlying variance. It contains a large number of 60 model terms, including main effects, interactions and quadratic effects, where the p-values indicate a high importance. Due to this high number of terms, an additional analysis is needed to extract the importance variables. At this point, as we are not able to validate the results in terms of their usefulness, we decided not to conduct any further analysis.
- The random forest model has a mean of squared residuals of 0.129 and explains 71.44 percent of the variance. The mean decrease in MSE is shown in table 3 and compared to the results of *Kriging*.
- The estimated activity parameters (theta) are shown in table 3.

Table 3: Variable importance

Weight	RF Mean MSE Decrease	Kriging theta values
1	117.31366	3.82746
2	9.83143	1e-04
3	59.79332	1.15356
4	17.86245	0.4643613
5	11.06656	0.2711343
6	14.05440	0.001178616
7	64.32869	2.221088
8	64.86968	1.164819
9	34.63966	1.8983
10	47.50249	3.45285

Table 3 indicates, that at least the most important weight (No. 1) and the least important weight (No. 2) are the same for both the *Kriging* and *random forest* model. The other weights also show some correlation. The

variable importance comparison shows, that the models are able to extract knowledge beyond the best found parameter setting. To validate the given results, we will need to use a designated experimental design, which will be part of future research.

Computation Time Comparison

An important aspect of every optimization technique is the total computation time. Table 4 shows approximated values for the metaheuristics and the surrogate-assisted optimization with the respective models. As the problem itself has nearly no computation time, the indicated values are mainly caused by the optimization algorithms. As the values indicate, surrogate-assisted optimization is in comparison very expensive. The model fitting, updating and optimization process is computationally expensive, particularly for a higher number of samples. This is especially visible for Kriging, which is very sensitive to higher sample sizes.

Table 4: Algorithm Computation Time. All values are approximated.

Algorithm	No. Evaluations	Computation Time
S-Ring Problem C	1	< 0.001 seconds
S-Ring Problem R	1	< 1 second
Metaheuristics	100	0.1 second
Metaheuristics	1000	1 second
Metaheuristics	1_{e+5}	1-2 minutes
surrRF	100	1 minute
surrSO	100	4 minutes
surrKR	100	8 minutes
surrRF	1000	1 hour
surrSO	1000	4 hours
surrKR	1000	> 1 day

At this point, we also have to consider that the *SPOT* implementation is a R-framework, which is in terms of computation time much inferior to

C or C++ based implementations. For instance, a re-implementation of the S-Ring simulator in R, which is normally implemented in C, is about 1000 times slower. We can assume, that an optimized version would be significant faster. Moreover, *SPO* performs sequential optimization, while the metaheuristics are able to conduct parallel evaluations.

Conclusion

In accordance to our hypotheses, the results show that surrogate-assisted optimization is a beneficial approach for the underlying NN control optimization task. The tested algorithms were capable of outperforming metaheuristic optimization methods. Furthermore, the surrogate can be used for significance analysis of the inputs and weighted connections to further exploit problem information. We can thus assume that surrogate-assisted optimization is able to provide a greater understanding of the learning process. The clear downside of the surrogate-assisted optimization is the large computation time, which is more than 10000 times larger than these of metaheuristic optimization. However, this huge downside becomes less significant in scenarios where the objective function evaluations become very expensive, e.g., in the area of several minutes. The model fitting and optimization process could be conducted simultaneously to the real-time evaluations. Also, not considered here are optimized and parallel surrogate-assisted approaches, which could considerable improve the computation time. In this study, we used the default parameters for all given algorithms, whereby no extensive research was made to optimize the *SPOT* default parameters, while the metaheuristic implementations default parameters are commonly optimized or include self-adaptive procedures to show comparable results. An extended study to identify generally good settings for large set of problems could be helpful to further improve general performance. In future research, we will test the applicability to a larger range of difficult problems from the area of artificial intelligence and evolutionary robotics. Moreover, we will study the usefulness of the extracted variable importance for evolutionary robotics.

Acknowledgements

This work is part of a project that has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement no. 692286.

References

[1] T. Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[2] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 773–780. IEEE, 2005.

[3] T. Bartz-Beielstein, M. Preuss, and S. Markon. Validation and optimization of an elevator simulation model with modern search heuristics. *Metaheuristics: Progress as Real Problem Solvers*, pages 109–128, 2005.

[4] J. C. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[5] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[6] Á. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3(2):124–141, 1999.

[7] O. Flasch, T. Bartz-Beielstein, A. Davtyan, P. Koch, W. Konen, T. D. Oyetoyan, and M. Tamutan. Comparing ci methods for prediction models in environmental engineering. In *Proc. of CEC*, 2010.

[8] A. Forrester, A. Sobester, and A. Keane. *Engineering Design via Surrogate Modelling*. Wiley, 2008.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[10] A. Liaw and M. Wiener. Classification and Regression by randomForest. *R News*, 2(3):18–22, 2002.

[11] W. R. Mebane Jr and J. S. Sekhon. Genetic optimization using derivatives: the rgenoud package for r. *Journal of Statistical Software*, 42(11):1–26, 2011.

[12] K. M. Mullen. Continuous global optimization in r. *Journal of Statistical Software*, 60(6):1–45, 2014.

[13] K. M. Mullen, D. Ardia, D. L. Gil, D. Windover, and J. Cline. Deoptim: An r package for global optimization by differential evolution. 2009.

[14] Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*, 41(4):687–696, 2003.

[15] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

[16] L. Scrucca. Ga: a package for genetic algorithms in r. *Journal of Statistical Software*, 53(4):1–37, 2013.

[17] J. S. Sekhon and W. R. Mebane. Genetic optimization using derivatives. *Political Analysis*, 7:187–210, 1998.

[18] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[19] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[20] Y. Xiang, S. Gubian, B. Suomela, and J. Hoeng. Generalized simulated annealing for global optimization: The gensa package. *R Journal*, 5(1), 2013.