Exploring and Improving the GT-SUITE Genetic Algorithm

Ryan Dudgeon

December 2020

1. Summary

Computer-aided engineering (CAE) software enables virtual testing of a variety of physical components, products, and systems so that they can be tested and optimized more quickly and cheaply compared to physical testing. An automated optimization tool is often needed to find the best set of input values to achieve model calibration and engineering design objectives for complex, non-linear problems involving many inputs. Genetic algorithms are common search algorithms used for solving these difficult problems. The genetic algorithm in GT-SUITE can often be slow to converge and exhibit a large variance in outcomes, so its exploitation attribute has been enhanced by incorporating metamodeling. The improved algorithm was tested on 25 mathematical and physics-based test problems, and its performance was measured against the standard genetic algorithm. For almost all test models, the improved algorithm found better response values in fewer total design iterations and decreased the variance in performance. This paper also provides background information on the genetic algorithm and explores the relationship between the population size and number of generations.

2. Introduction to Model-Based Optimization and Search Algorithms

Mathematical optimization is the search for optimal model input parameters to minimize or maximize one or more of its predicted outputs. This paper will refer to model input parameters as factors, and predicted model outputs as responses. Optimization is a powerful tool to the modeler for two main tasks, both of which can enhance the modeler's understanding of the relationship between factors and responses, and therefore the model in general. The first task is model calibration, which is the process of tuning model inputs, whose values have a reasonable degree of uncertainty, to get one or more desired model outputs to align better with measurement data, thereby achieving a more accurate model. The second task is engineering design, which is the process of finding optimal design parameters for which the design engineer is responsible, for the purpose of improving a real component, product, or system.

To avoid the rote task of simulating every possible combination of factor values with an acceptable degree of resolution and within their specified bounds, an optimizer employs a search algorithm to more efficiently find the optimal solution in as few model evaluations (referred to as design iterations) as possible. A branch of computer science is dedicated to the study and development of different search algorithms and strategies, and one can find multiple hundreds of different algorithms in the literature, as well as hybrid algorithms that combine characteristics of two or more.

Optimization search algorithms can be categorized into two groups: local and global. Local methods rely on the response obtained from the initial factor values, as well as responses obtained around the initial values, to decide how to further change them. One category of local methods includes those that calculate or evaluate the slope of the response with respect to the factors, but not all local methods require evaluating the slope. A downside of local methods is that they are susceptible to converging to a local optimum, and not the global optimum, for multi-modal systems as shown in Figure 1. Even an algorithm such as Nelder-Mean Simplex, a local, black-box method that does not rely on slopes and which tends to be a robust method, is susceptible to converging to a local optimum.

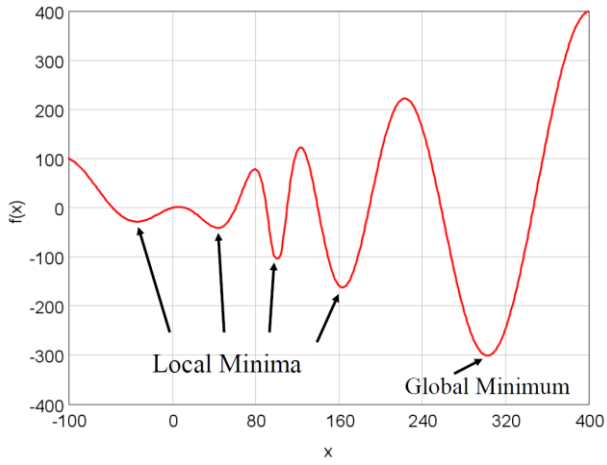


Figure 1: A multi-modal response vs. a single factor.

Global search algorithms include stochastic methods to more thoroughly search the design space, which is the multi-dimensional region between the lower and upper bounds of the factors. This characteristic is referred to as exploration and gives global methods a much higher likelihood of finding the global optimum for a multi-modal system. At the same time, global search algorithms also include methods to facilitate exploitation, which is the refinement or local search for better solutions given the better ones that are already known. A global search algorithm balances the effort given to exploration and exploitation, but one generally desires more exploration at the beginning of an optimization run and more exploitation near the end. Global search algorithms are generally slower to converge than local methods, and global methods are still susceptible to converging to a local optimum. No optimization algorithm can guarantee finding the global optimum for complex problems, and it is impossible for a search method to inform the user that a global optimum has been found.

GT-SUITE's genetic algorithm is NSGA-III, which is a non-dominated sorting algorithm. It was introduced in [1] and implemented via the MOEA Framework, which is a free and open-sourced Java library of evolutionary optimization algorithms [2]. NSGA-III is an extension to NSGA-II, which was introduced in [3] in 2002 and has been widely used throughout the optimization community.

A genetic algorithm mimics biological evolution by applying mathematical equivalents of selection, cross-over, and mutation to a population of agents over multiple generations. An agent is a specific combination of factor values, thereby representing one design iteration. A key genetic algorithm input is the population size, which, after getting initialized with a set of agents, proceeds to evolve from generation to generation. The total number of model evaluations or design iterations is the population size multiplied by the number of generations. A simplistic pseudocode is given here.

```
Generate the initial population of N agents using Latin Hypercube sampling and set G = 0
Evaluate the population
While G < Number of Specified Generations:
   Apply selection, cross-over, and mutation to the population
   Evaluate the new population
   Increment G = G + 1
```

3. Evaluating Optimization Search Algorithms

Optimization algorithms must be tested and evaluated using a wide range of different models with varying complexities and numbers of factors. The optimization community has adopted many different mathematical test functions that can be tested on any optimization tool. The GT-SUITE genetic algorithm was evaluated (and improvements were tested) using 7 mathematical test functions and 10 physics-based models. Most of the models were chosen for their high level of complexity of the optimization problem, although a few models were used that consist of simpler optimization problems. Many mathematical test functions can accept any number of input variables, thereby creating optimization problems with any number of input factors. For example, the Rastrigin function is the following, where *x* is a factor and *i* is the factor number for *n* total factors:

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right]$$

Four of the mathematical test functions were reused with different numbers of input factors. For example, the Rastrigin function was used three times, with 5, 10, and 20 factors. The number of factors was chosen based on the number of factors that are commonly used in the context of optimization with GT-SUITE. For example, it is much more common for GT-SUITE optimization models to utilize a lower number of factors, for example 3-6. It is less common, but not unusual, to need to optimize a number of factors around 10, and it is much rarer, although not unprecedented, to need to optimize as many as 20 factors. Each mathematical function was implemented in a GT-SUITE model. Table 1 lists the 15 models that evaluated mathematical functions.

Model Name	Number of Factors
Ackley	2
Eggholder	2
Quadratic-4Factors	4
Rastrigin-5Factors	5
Rastrigin-10Factors	10
Rastrigin-20Factors	20
Rosenbrock-5Factors	5
Rosenbrock-10Factors	10
Rosenbrock-20Factors	20
Schwefel-5Factors	5
Schwefel-10Factors	10
Schwefel-15Factors	15
ST-5Factors	5
ST-10Factors	10
ST-15Factors	15

Table 1: Mathematical models used for evaluation and testing.

For physics-based models, it is often not possible to identify or evaluate the mathematical relationship between input factors and the response of interest because multiple different physical processes can occur between model inputs and outputs, however the relationship is often known to be highly non-linear. Table 2 lists the 10 physics-based GT-SUITE models that were tested. Because many GT-SUITE users rely on optimization for combustion calibration and aftertreatment kinetic calibration, three models from each category were included. The combustion calibration models are DIPulse-A, DIPulse-B, and SITurb. The aftertreatment models are DOC, PNA, and SCR. All models, both mathematical and physics-based, were minimization optimization problems. Appendix A contains more details about the test models.

Model Name	Number of Factors
DIPulse-A	4
DIPulse-B	7
DOC	10
Engine	9
EngineKnock	21
GPF	7
Muffler	3
PNA	5
SCR	5
SITurb	4

Table 2: Physics-based models used for evaluation and testing.

The GT-SUITE genetic algorithm, like other evolutionary algorithms, is a stochastic method that incorporates randomness in two ways. First and most importantly, the creation of the initial population uses Latin Hypercube sampling to choose initial agents within the search space, and Latin Hypercube sampling uses randomness. Second, the evolutionary processes of selection, cross-over, and mutation also rely on randomness. These sources of randomness are controlled by selecting a random seed for each run. An entire genetic algorithm run can be deterministically reproduced by using the same random seed, which can be directly entered in the GT-SUITE model.

Because of these sources of randomness, one cannot adequately evaluate an evolutionary algorithm by running it a single time on a given model. Because it is not deterministic, each run of the genetic algorithm will proceed differently, and depending on the settings, the variance in final outcomes could be significant. Running the optimization multiple times will result in a distribution of final outcomes.

Figure 2 displays histograms showing the best achieved minimum response after running five optimization models 100 times, which we call trials. The x-axis shows the variance in performance, and an important detail about this plot is that the optimizations were not run with the intent to achieve a specific level of convergence. An important conclusion from Figure 2 is that the genetic algorithm performance is often not normally distributed, and therefore using the *average* response over multiple optimization runs may not be an appropriate metric for evaluating performance.

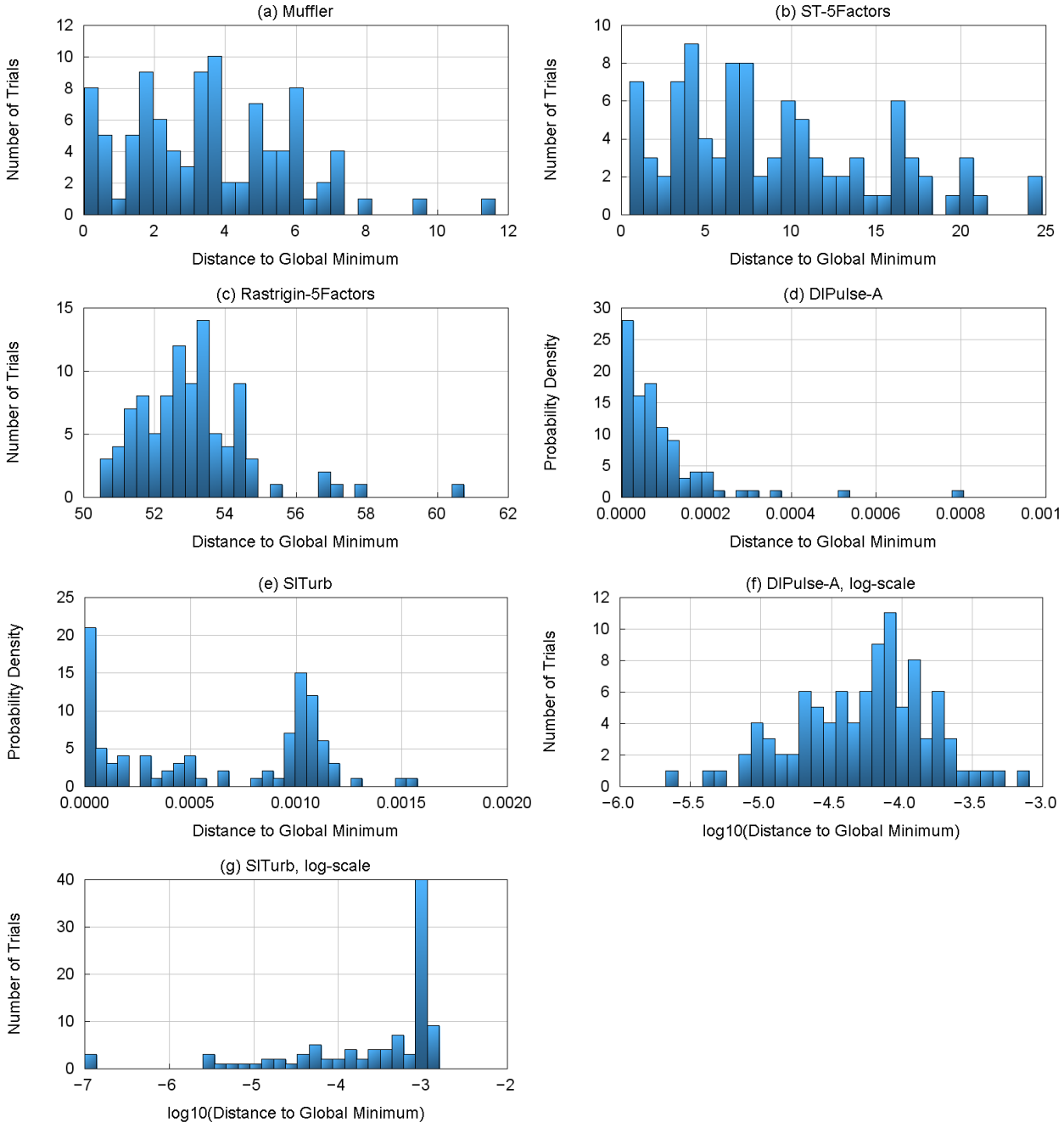


Figure 2: Histograms of the best achieved minimum for 100 trials for four different models. Two models whose best achieved minima are less than zero are duplicated with a log scale, such that plots (d) and (e) convey the same information as (f) and (g), respectively.

Running the genetic algorithm multiple times on the same model and with the same algorithm settings (hyperparameters), each time with a different random seed, is necessary to evaluate the algorithm. For this study, 20 trials were run on each model for a given algorithm and set of hyperparameters.

The performance of an optimization run generally consists of viewing the change in the response variable plotted against the design iteration number. To evaluate optimization performance, we introduce the

term *progress*, which is defined as the best response value at any given design iteration. Figure 3 shows a scatter plot for an arbitrary optimization run and its corresponding progress.

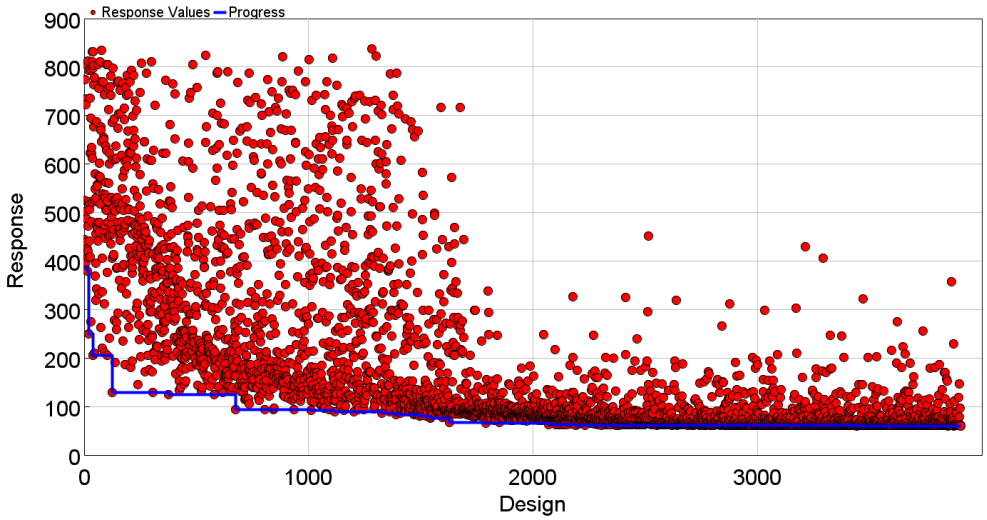


Figure 3: Response (red) vs. design iteration number and the resulting progress (blue) for an arbitrary optimization run.

Furthermore, for each test model, one or more exhaustive optimization runs were run to determine the best possible solution (global minimum) with a high precision of significant digits. Each test model was then modified to output the distance to the global minimum instead of the nominal response value.

Distance to Global Minimum = Predicted Response – Global Minimum

This modification allows easy evaluation and visual comparison of multiple optimization runs. Regardless of the global minimum and magnitude of the response, any two progress values can be directly compared. For example, a distance of 200 is twice as good as a distance of 400, and a distance of 0.001 is four times as good as a distance of 0.004. This modification also facilitates easier comparison of the precision of multiple optimization runs as the response value approaches the global minimum.

Figure 4 shows the effect of this modification, where the plot on the left uses the nominal response value on the y-axis, and the plot on the right uses the distance to the global minimum and is log-scale. Although the final response values at design 900 for the two progress lines appear identical in the plot on the left, the plot on the right shows that “Search Algorithm 2” is able to find a solution that is about 3 times better and more precise than “Search Algorithm 1” (1.5E-5 vs. 4.5E-5). Unless otherwise noted, this paper will use the terms *response* and *distance to global minimum* interchangeably.

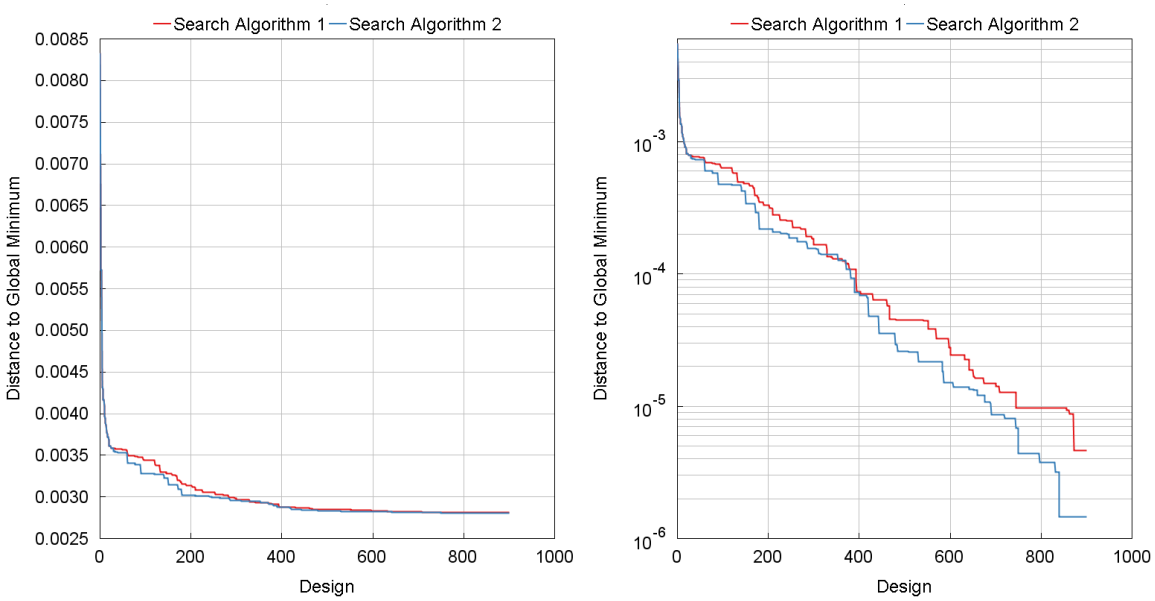


Figure 4: Performance comparison of two optimization runs using nominal progress (response) values on the left and the distance to the global minimum and a log-scale y-axis on the right. The global minimum is 0.0028.

Because the performance of multiple optimization runs on the same model may not conform to a normal distribution, results from multiple runs were combined to a single metric by calculating the median progress value, which is calculated at every design iteration as illustrated in Figure 5.

Aside from median progress and the final best achieved median progress value, another metric used to evaluate genetic algorithm performance was maximum progress. Maximum progress refers to taking the maximum response value among the multiple trials at each design iteration. The maximum progress is a measure of the worst performance that can be expected and provides an estimate of the right tail of the distribution of progress depicted in Figure 2.

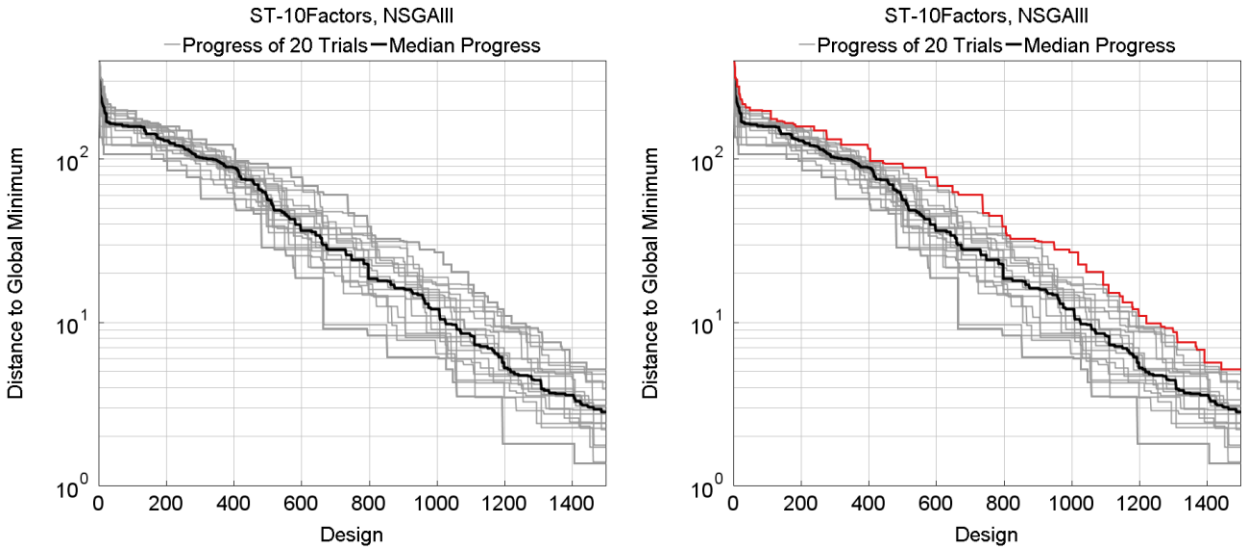


Figure 5: Progress of 20 optimization trials (gray) performed on the same model, along with the median progress (black). The plot on the right shows the maximum progress in red.

4. Exploring the Genetic Algorithm

This section explores the performance of the genetic algorithm and impacts from its hyperparameters. The two most important hyperparameters are population size and number of generations. The population size is the number of agents in the population that evolves over the generations. A larger population size provides a larger extent of sampling for the initial population, thereby enabling a greater likelihood for finding the global optimum for difficult problems. A larger population size also contributes to maintaining high diversity among agents in the population, where diversity refers to agents being sufficiently different from each other. As the number of factors increases, thereby increasing the number of dimensions in the search space, a larger population size is generally needed to improve the likelihood of finding good solutions.

The number of generations determines the number of times the population evolves. With more generations, the population "turns over" more, and the algorithm has more opportunity to learn the system and adapt itself. An optimization problem generally needs 10-20 generations *at minimum*, and many problems will require many more.

The total number of design iterations that the optimizer runs is determined by the population size multiplied by the number of generations. If finding a very good (or the best) solution with a high degree of certainty is important, the user can simply increase both the population size and number of generations, but of course, increasing both simply leads to a large number of model evaluations to run. There is a trade-off between these two settings, where a smaller population size generally provides faster convergence because, given a set number of design iterations, more generations are run.

This concept is demonstrated by the Quadratic-4Factors model, where it was run for 1200 design iterations with population sizes of 10, 20, 30, and 40. Each line in Figure 6 represents the median progress for the population size (as a reminder, a median progress line results from running 20 trials). Figure 6 shows a clear advantage for using the smallest population size of 10, where a better optimum is found far more quickly than when using the larger population sizes. For example, a response value of 0.01 was found after about 500 design iterations when using a population size of 10, but a population size of 20 required about 725 design iterations to obtain the same response value. After 1200 designs, a population size of 10 finds a response value of about 0.0007, whereas a population size of 40 has only found a response value of about 0.01. The difference in performance is attributed to 120 generations (=1200/10) being used for a population size of 10, whereas only 30 generations (=1200/40) had been run with a population size of 40.

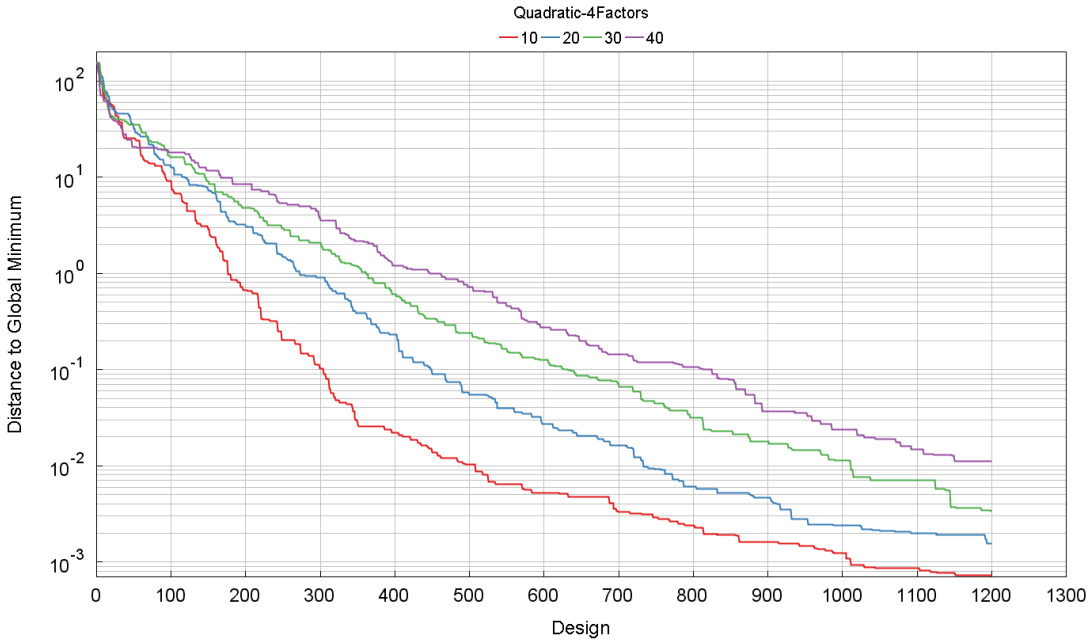


Figure 6: Median progress with the Quadratic-4Factors model with different population sizes. The number of generations was adjusted to obtain 1200 total design iterations.

However, smaller population sizes lead to lower population diversity, which leads to higher probability of converging to an optimum that is not the global one, and not every model is expected to exhibit a similar trend as Figure 6. For example, Figure 7 shows the performance vs. population size for two more complex models, ST-5Factors and Schwefel-5Factors. Although smaller population sizes still result in faster convergence, if the population size is too small, the algorithm gets stuck in a local optimum, as is the situation with a population size of 10 for ST-5Factors. For Schwefel-5Factors, population sizes of 10 and 20 lead to convergence to a local minimum without the ability to make further progress.

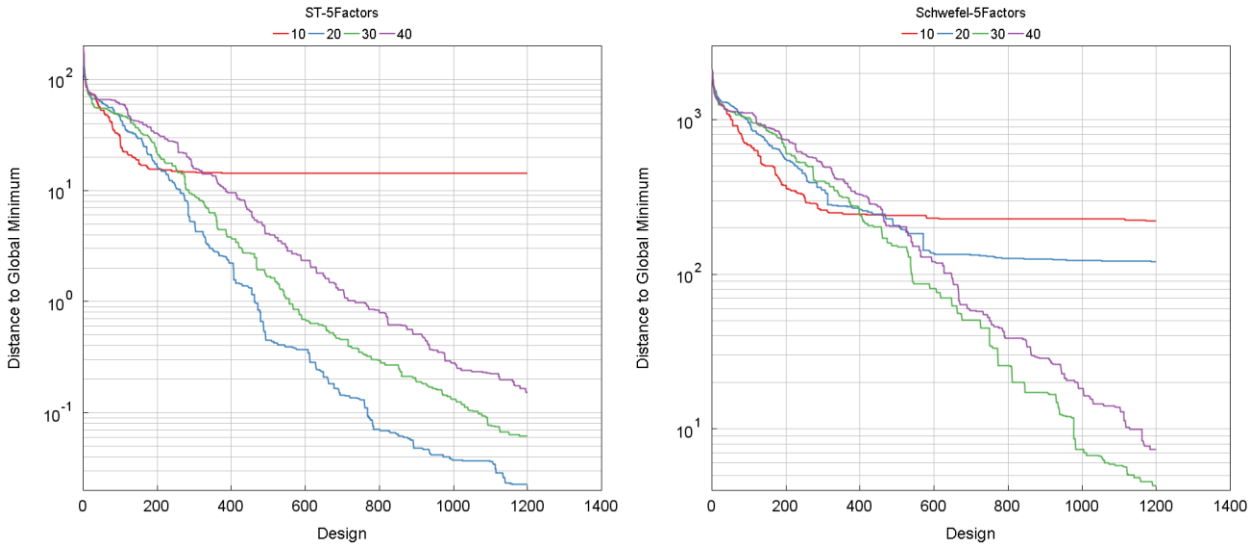
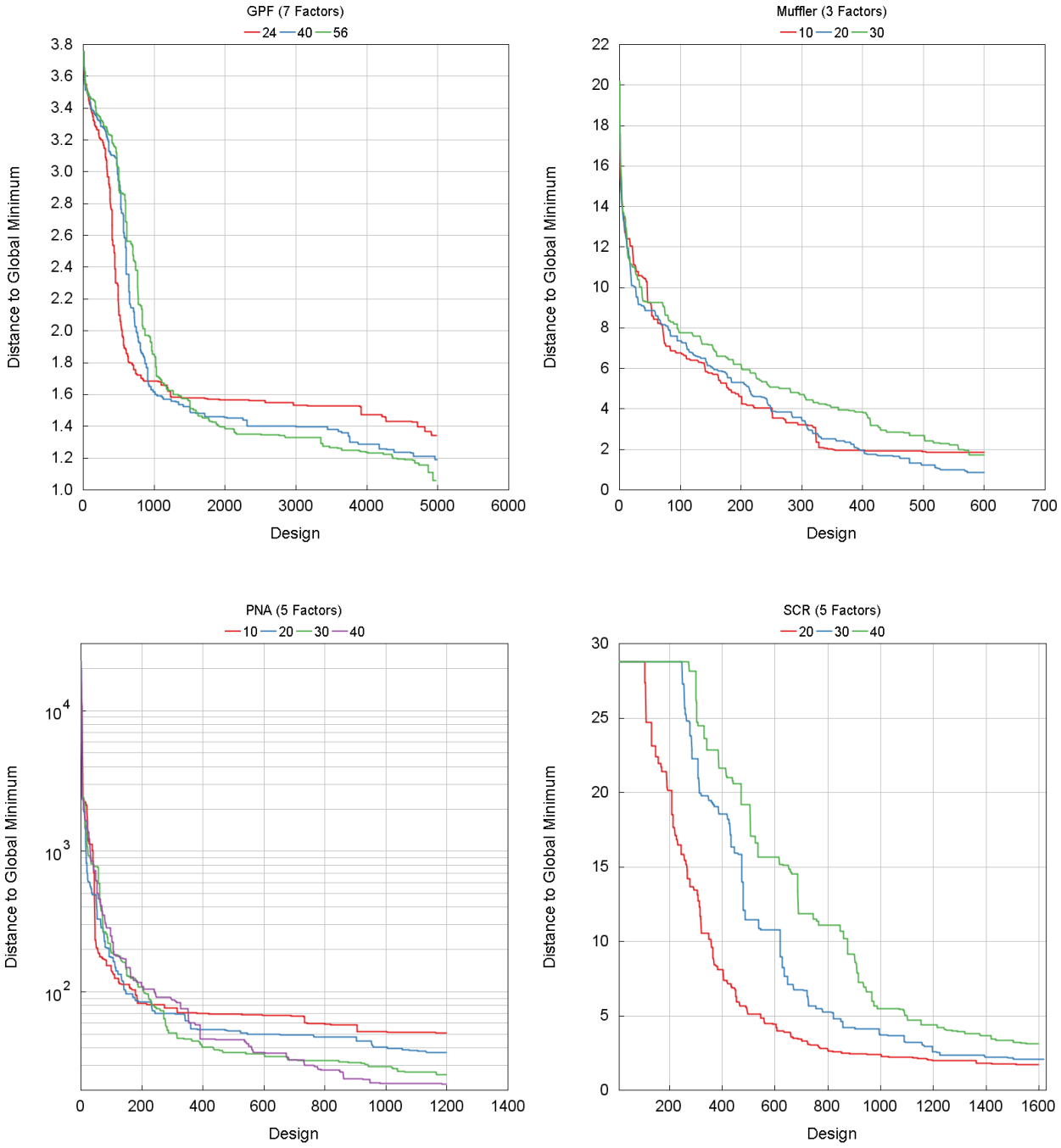


Figure 7: Median progress for the ST-5Factors and Schwefel-5Factors models with different population sizes. The number of generations was adjusted to obtain 1200 total design iterations.

Figure 8 displays the relationship between population size and number of generations for 10 other test models. In particular, the GPF and PNA models exhibit faster early progress for lower population sizes, but larger population sizes provide better progress at the later stages of the optimization run.



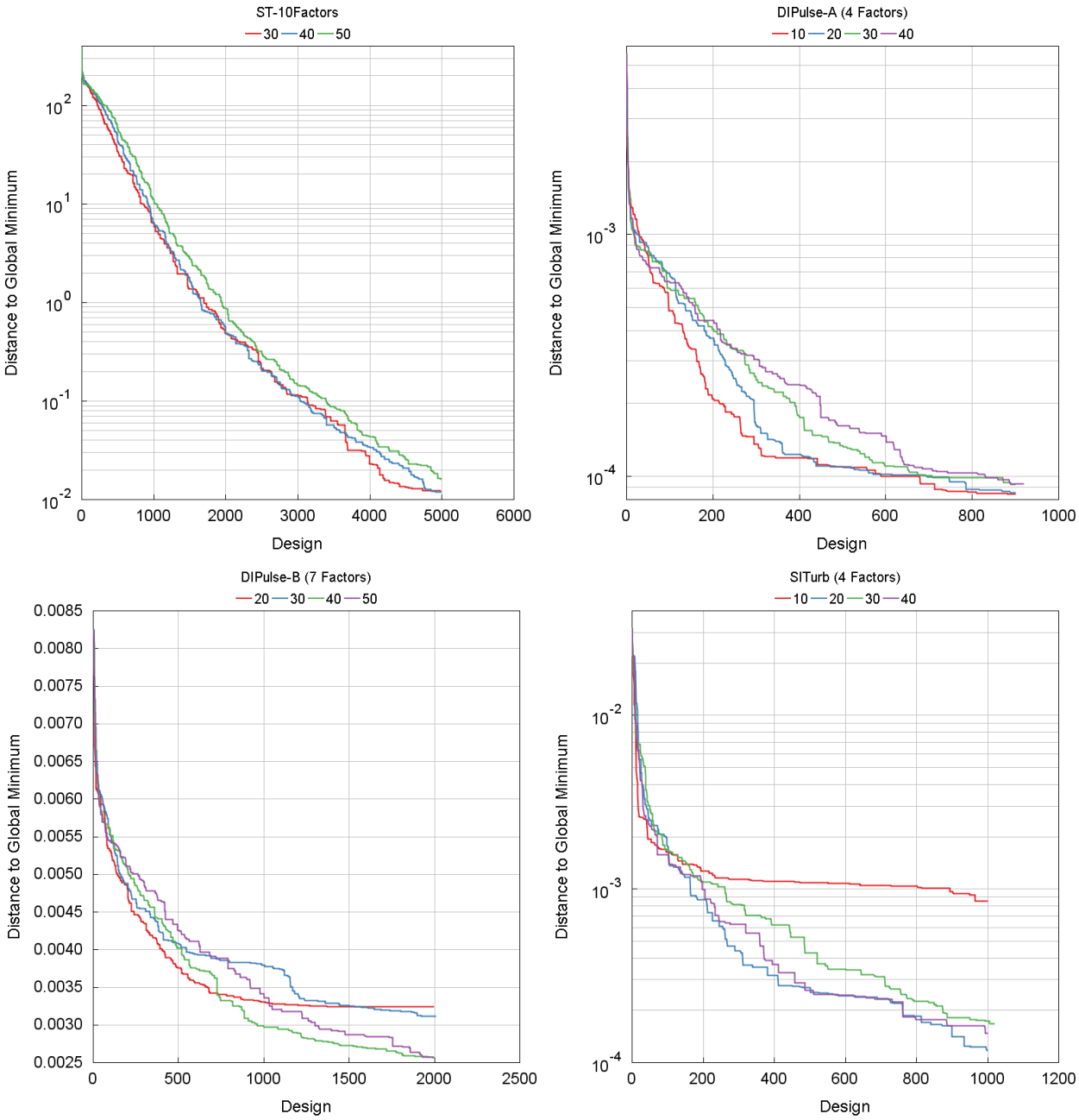


Figure 8: Median progress for eight additional models with different population sizes.

5. Improving the Genetic Algorithm

The genetic algorithm (referred to as the standard GA) was improved in v2021 GT-SUITE by incorporating metamodeling, and the modified search algorithm is referred to as the Accelerated GA. Also referred to as a response surface or surrogate model, a metamodel is a mathematical representation of response data that uses factors as input variables. The known data points are used to train the metamodel for the purpose of predicting response values between the known points.

The metamodel type used to improve the genetic algorithm is Kriging, which is a data-driven, Gaussian interpolation method that captures the nonlinear behavior of complex data sets. Within the assumptions of its covariance priors, Kriging provides the best linear unbiased prediction between known data points.

A common graphic illustrating the method in one dimensional is shown in Figure 9 [4], which shows 7 data samples (independent variable in blue and dependent variable in red), the normally distributed confidence intervals (99% in this example) in gray, and the Kriging prediction in represented by the green line, which passes through the mean of the confidence intervals.

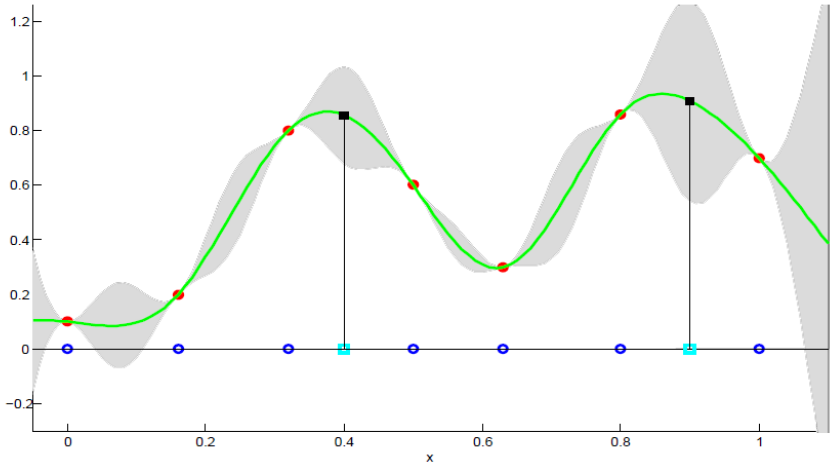


Figure 9: Kriging interpolation example in one dimension using seven data samples (independent variable in blue and dependent variable in red) [4]. The green line represents the Kriging prediction, which passes through the mean of the normally distributed confidence intervals (99% in this example) indicated by the gray shading. The black points are two highlighted predictions at the independent variable values of the cyan points.

The Accelerated GA works as follows. At the end of each genetic algorithm generation, the available data is trained to a Kriging metamodel. If the Kriging metamodel fits the data well, an optimization (independent from the main optimization) is run on the metamodel, which results in a set of factor values that represents the best approximation for where the minimum or maximum response value lies. This intermediate optimization run is computationally fast because it is only operating on the metamodel, and its results are not reported in any way to the user; it happens entirely behind the scenes. The resultant set of factor values are fed back to the main optimization, where they constitute an agent in the next population for the next generation. In other words, a real solver-based simulation will be run with the factor values that approximate the optimal response value. The rest of the population is created using the standard GA operations consisting of cross-over, mutation, and selection. If the Kriging metamodel does not fit the data well, then no optimization is run on the metamodel, and the entire new population will be created from standard genetic algorithm operations. The simple pseudocode from Section 2 is updated here.

```
Generate the initial population of N agents using Latin Hypercube sampling and set G = 0
Evaluate the population
While G < Number of Specified Generations:
  Create a Kriging metamodel using the available data
  If the Kriging metamodel is well trained:
    Perform an optimization on the Kriging metamodel
    Include the optimal solution in the new population
    Apply selection, cross-over, and mutation to determine the remaining N-1 agents
  Else:
    Apply selection, cross-over, and mutation to the population
  Evaluate the new population
  Increment G = G + 1
```

This method of utilizing metamodeling to approximate the optimal response value has been described in other work [5]. The method is essentially an enhancement to the genetic algorithm's exploitation attribute. It is not expected to improve the exploration ability of the algorithm, which is expected to already be very good.

The Accelerated GA was developed by running nearly 18,000 independent optimization runs, many of which consisted of testing different strategies for implementing metamodeling with the genetic algorithm. Once the strategy was decided, which took the form described in the previous paragraphs, many tests were run to identify the best hyperparameter values that worked best for the large set of test models. These hyperparameters include ones that are hard-coded in the Accelerated GA as well as the ones belonging to the existing genetic algorithm, such as cross-over rate, mutation rate, cross-over distribution index, and mutation rate distribution index. We will briefly mention that our testing found that no improvement was found by adjusting the four genetic algorithm hyperparameters for the Accelerated GA, so the default values in GT-SUITE remain the same as for the standard GA.

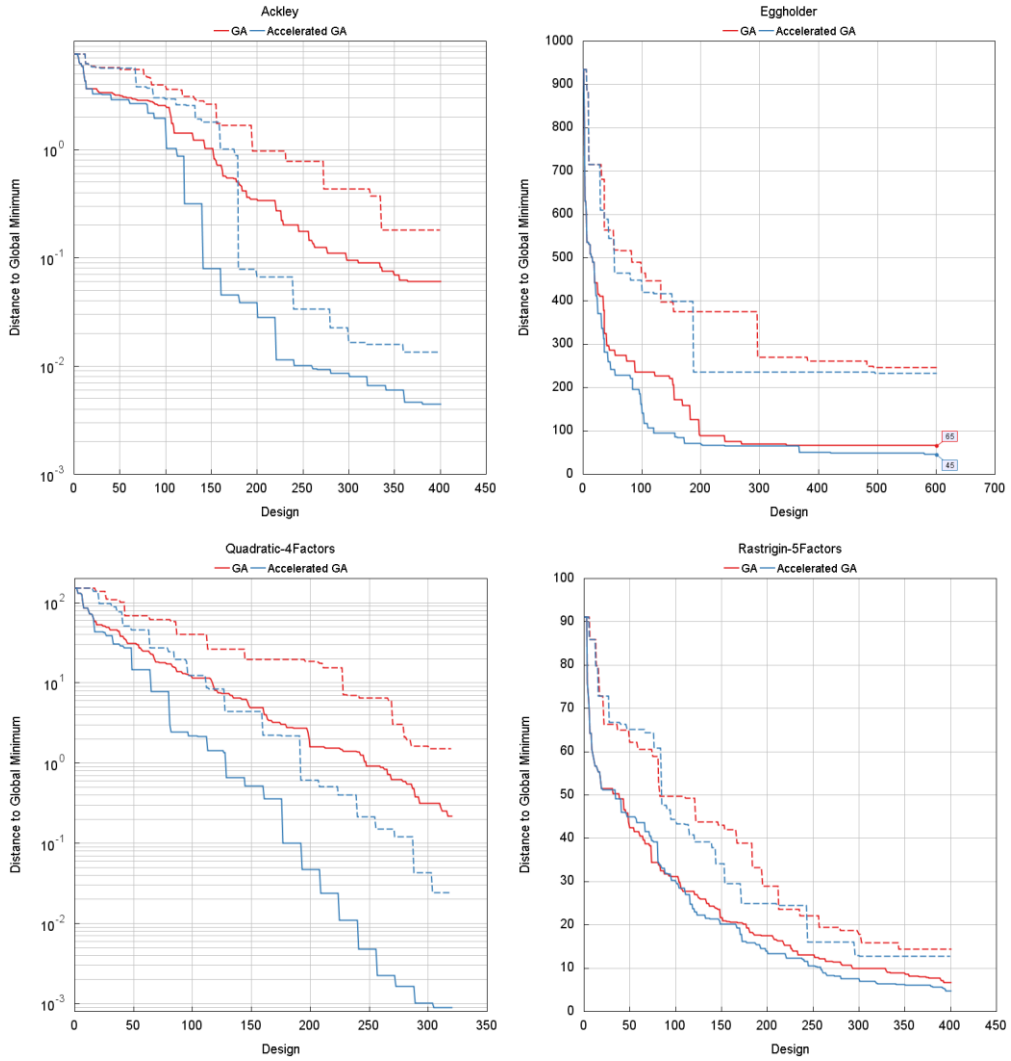
The optimization runs performed for testing always used the same 20 seeds to generate the same initial populations for a given model. In other words, for any given model, the 20 trials used for the final Accelerated GA results used the same initial populations as the 20 trials used for the standard genetic algorithm. The following section presents the final performance results between the two algorithms.

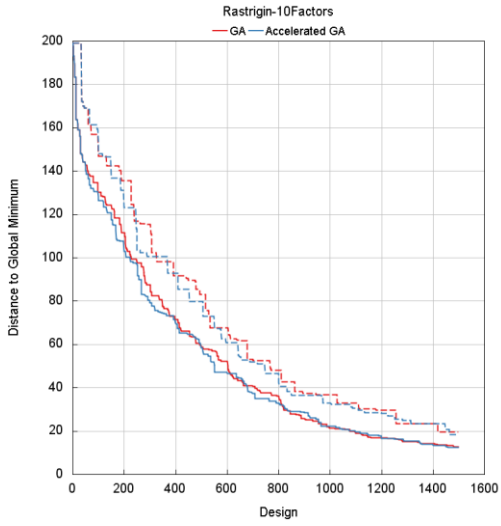
6. Evaluating the Accelerated GA

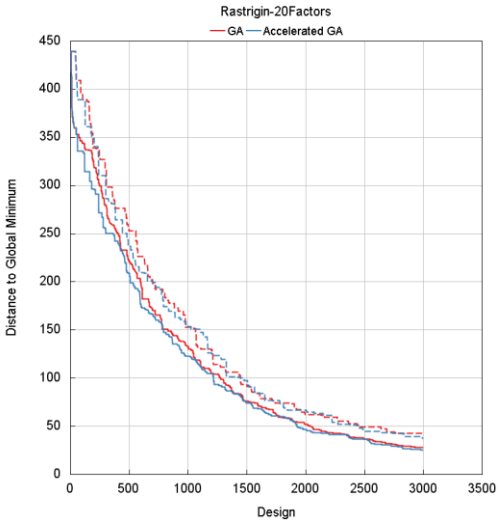
This section compares the performance between the standard GA and the Accelerated GA. Each plot displays a different model and includes the maximum progress in dashed lines. For clarification, there was no intention to run any given model to a certain level of convergence. The purpose of these plots is to compare general optimizer performance. The different models had some variation in the number of generations, which ranged from 20 for the simpler models to 50 for more complex ones. The population sizes were mostly chosen from the default values in GT-SUITE, which are dependent on the number of factors. However, some of them deviated from the default values based on some prior experience. For example, the combustion calibration models used population sizes of 30 or 40, even when the number of factors was only 4 (DIPulse-A).

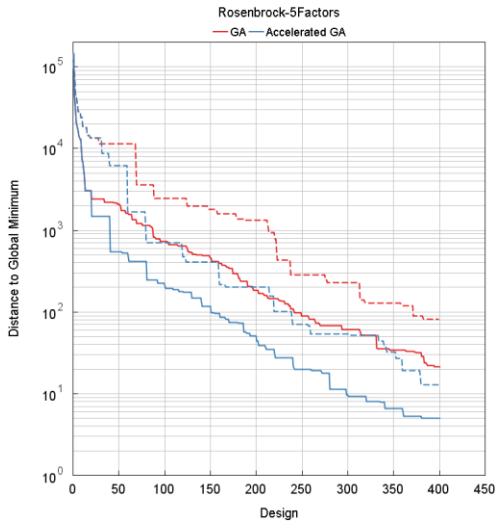
The comparisons for the mathematical test models are presented in Figure 10. The central objective for developing the Accelerated GA was to ensure that the performance was better or the same as that of the standard GA across all test models. Again, performance is mostly quantified as median progress and maximum progress, where smaller values for the latter translate to more consistency and less variance among the multiple trials. Therefore it is desirable to see the solid blue line lie below the solid red line, and it is also desirable to see the dashed blue line lie below than the dashed red line. These objectives

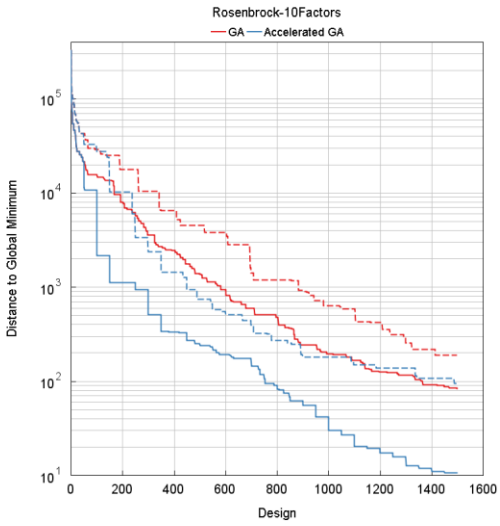
are mostly satisfied with the mathematical models. The Accelerated GA clearly benefits the Ackley, Quadratic, Rosenbrock, and ST functions. The advantage to using the Accelerated GA is small for the Eggholder and Schwefel models, and there is no advantage for the Rastrigin models. The only way in which the Accelerated GA performs worse than the standard GA is with the maximum progress of the Eggholder and ST models, meaning there is a little more variance in performance.

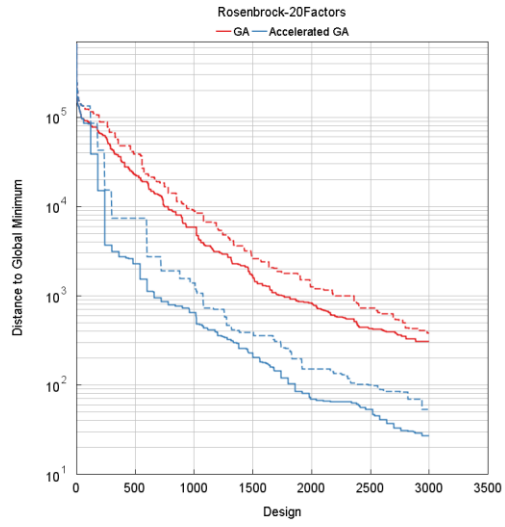


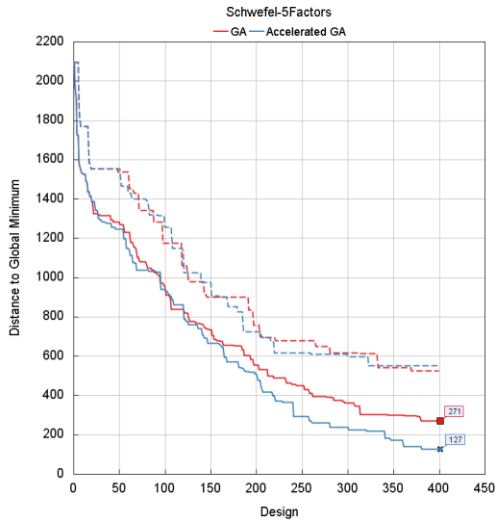

Rastrigin-10Factors
GA
Accelerated GA
Distance to Global Minimum
Design

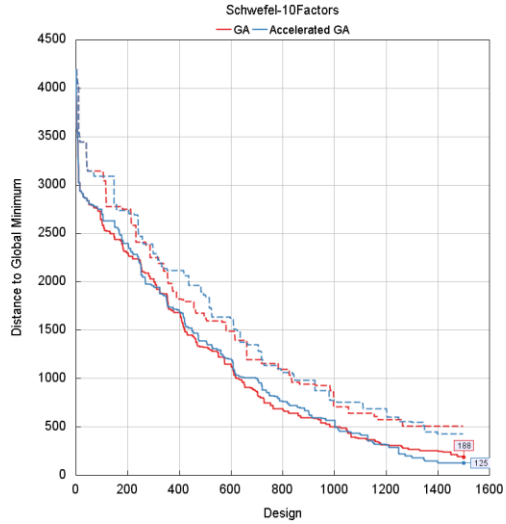

Rastrigin-20Factors
GA
Accelerated GA
Distance to Global Minimum
Design

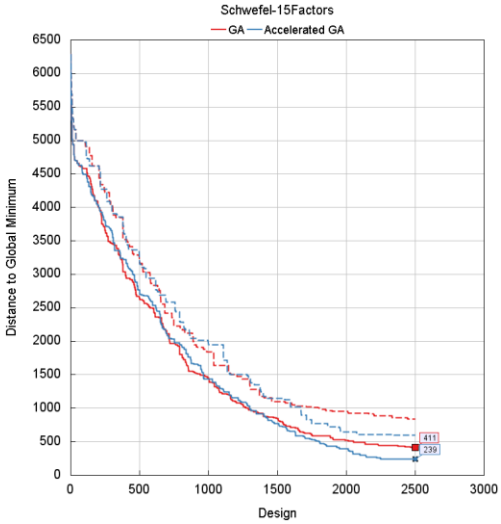

Rosenbrock-5Factors
GA
Accelerated GA
Distance to Global Minimum
Design

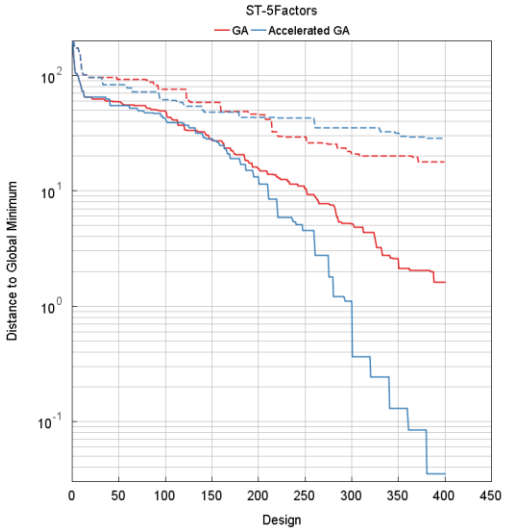

Rosenbrock-10Factors
GA
Accelerated GA
Distance to Global Minimum
Design

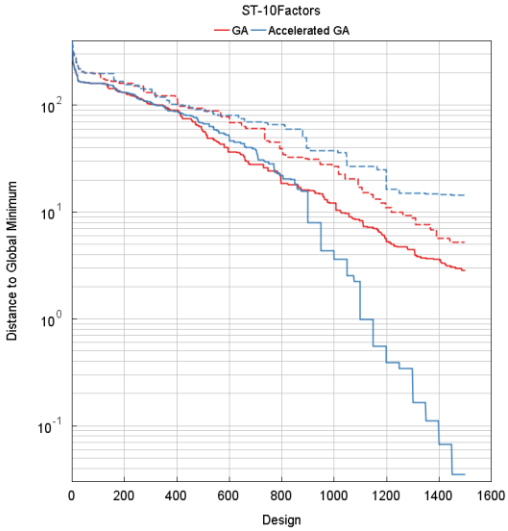

Rosenbrock-20Factors
GA
Accelerated GA
Distance to Global Minimum
Design


Schwefel-5Factors
GA
Accelerated GA
Distance to Global Minimum
271
127
Design


Schwefel-10Factors
GA
Accelerated GA
Distance to Global Minimum
Design
188
125


Schwefel-15Factors
GA
Accelerated GA
Distance to Global Minimum
Design
411
239


ST-5Factors
GA
Accelerated GA
Distance to Global Minimum
Design


ST-10Factors
GA
Accelerated GA
Distance to Global Minimum
Design

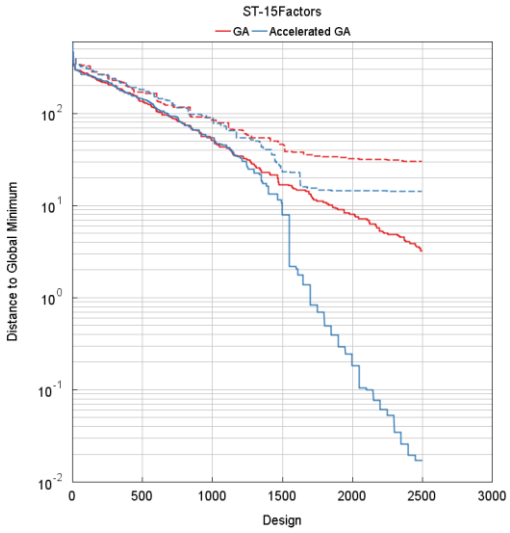
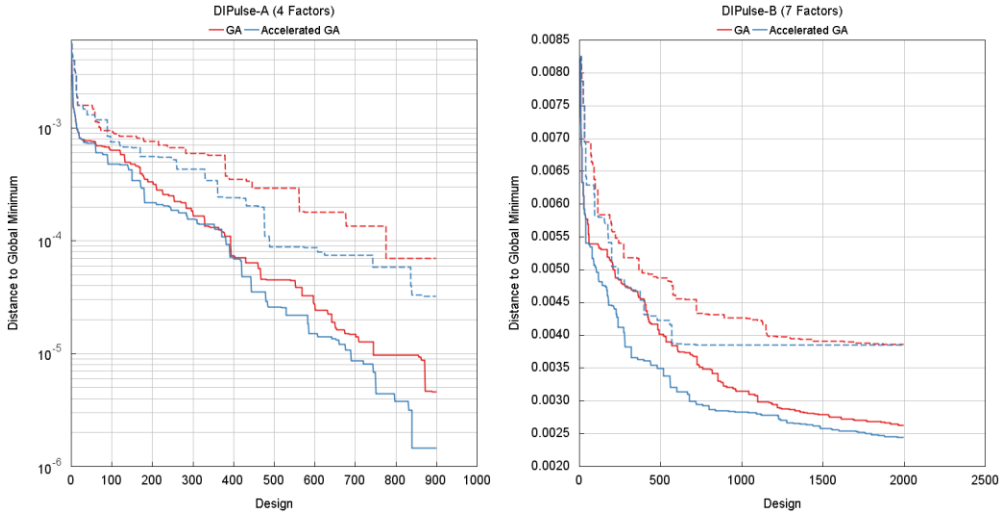
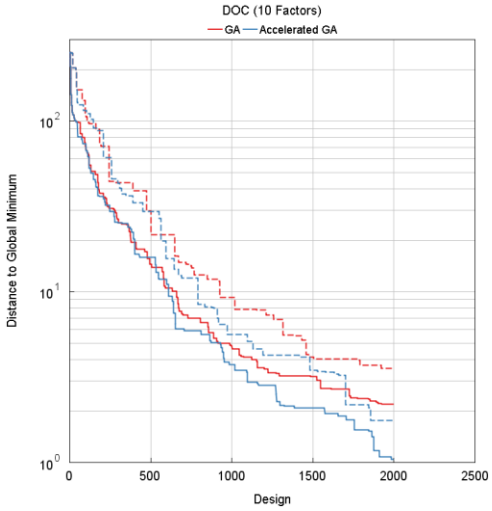

ST-15Factors
GA
Accelerated GA
Distance to Global Minimum
Design

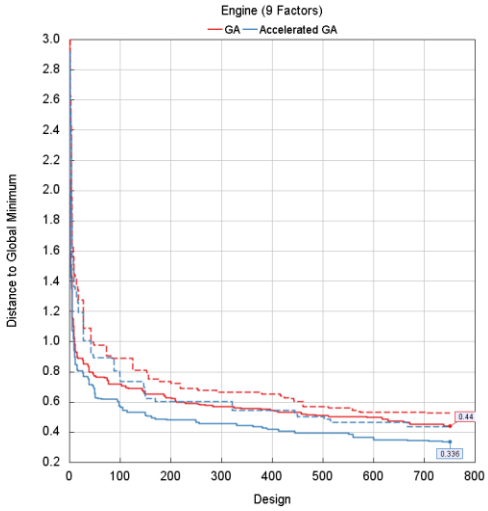
Figure 10: Comparison of median and max progress between the two algorithms for the mathematical test models.

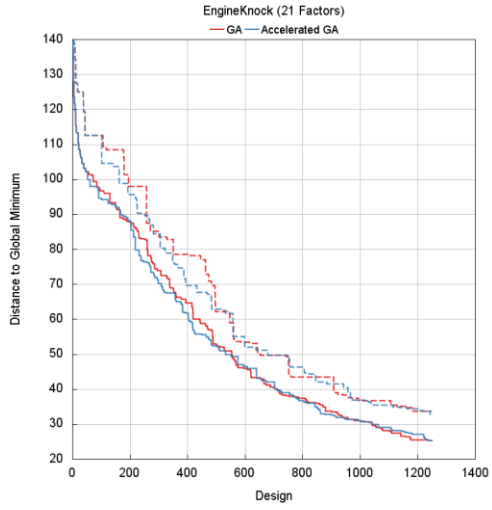
Another observation from Figure 10 is that there is not much benefit from testing the Rastrigin, Rosenbrock, Schwefel, and ST functions multiple times with different numbers of factors, as the relative comparison between the two algorithms is the same regardless of the number of factors used. In fact, this trend was consistent throughout testing; for example, many different strategies and hyperparameters for the Accelerated GA were tested, and whatever the comparison between any two tests turned out to be when using 5 factors, the comparison was always almost identical for the same mathematical function using 10, 15, or 20 factors.

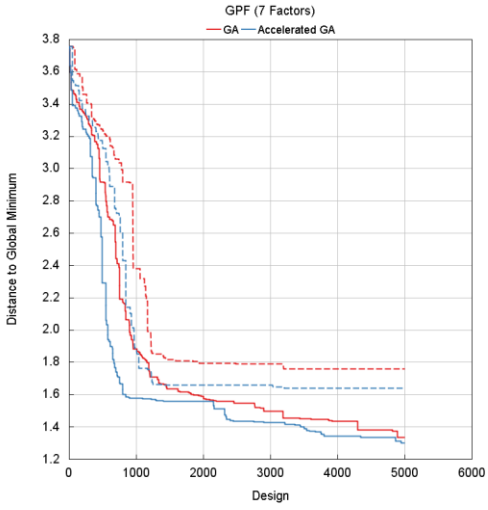
The comparisons for the physics-based models are presented in Figure 11. The Accelerated GA performs better than the standard GA for the three combustion calibration models (DIPulse-A, DIPulse-B, and SITurb). For DIPulse-B, the Accelerated GA finds better solutions multiple hundreds of design iterations sooner than the standard GA, as measured by the horizontal distance between the two median progress lines. The Accelerated GA performs worse than the standard GA for the Muffler model; although both algorithms converge to a local minimum and are unable to make further progress, the standard GA finds a minimum that is about twice as good as the Accelerated GA. Substantial benefit is gained for the DOC, Engine, and SCR models. For the GPF model, although the final best progress that is found by the two algorithms is the same, the Accelerated GA often achieves a given progress value much sooner than the standard GA, sometimes as many as 1000 designs sooner. The Accelerated GA provided no benefit to the EngineKnock and PNA models; however, these two models are particularly difficult optimization problems. EngineKnock consists of 21 factors and an optimization constraint, and for PNA, neither algorithm can substantially decrease the response to an acceptably low value. As Figure 8 shows, a larger population size and number of generations are needed to obtain better solutions.

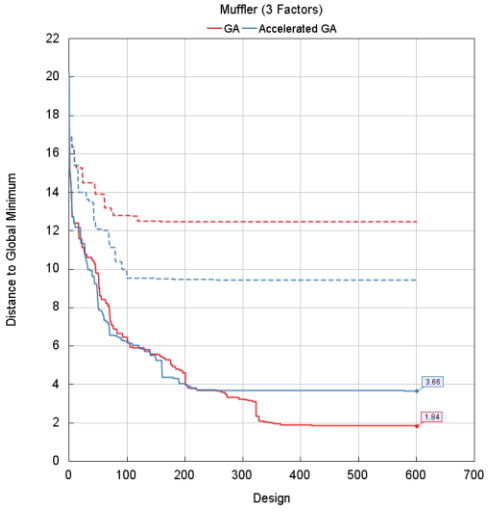


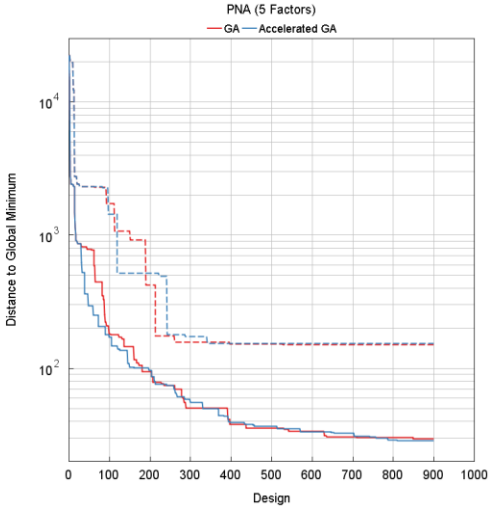

DOC (10 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design


Engine (9 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design
0.44
0.336


EngineKnock (21 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design


GPF (7 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design


Muffler (3 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design
3.66
1.84


PNA (5 Factors)
GA
Accelerated GA
Distance to Global Minimum
Design

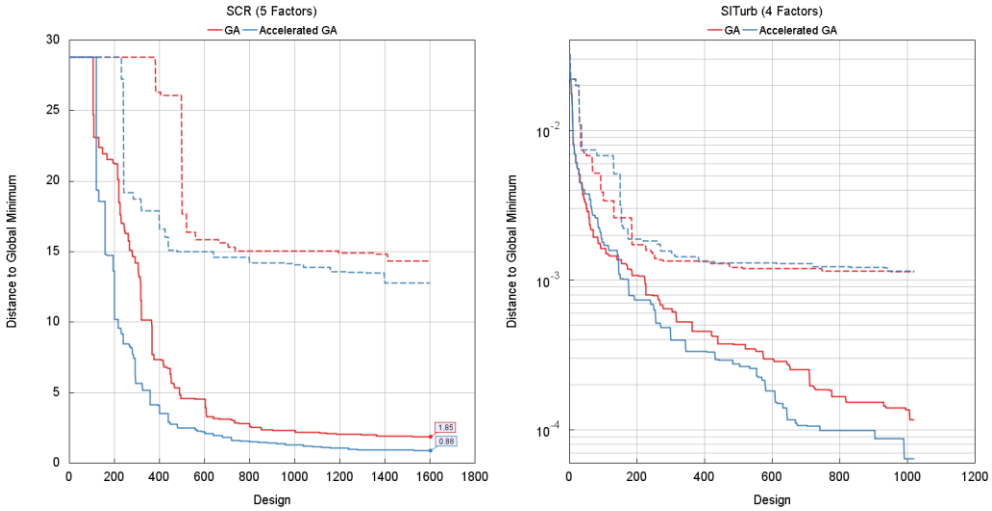


Figure 11: Comparison of median and max progress between the two algorithms for the physics-based test models.

We return to the Muffler model for which the Accelerated GA performed worse than the standard GA. Given that the Accelerated GA primarily improves the exploitation process of the genetic algorithm, and the population size is the primary knob for improving the exploration process, we experimented with increasing the population size for this model.

The population size for the Muffler model was doubled from 10 to 20, and the number of generations was cut in half to maintain the total number of design iterations. It is not very fair to simply compare the performance between the standard GA using a population size of 10 to the Accelerated GA using a population size of 20; it's still possible that the standard GA with an increased population size of 20 would perform just as well or better than the Accelerated GA with a population size of 20. Figure 12 therefore displays the progress for both algorithms using both population sizes. Figure 12 shows that with the larger population size, the standard GA still finds a better final response than the Accelerated GA. However, the Accelerated GA with population size 20 performs better for the first 520 design iterations, and the final response is not as different from that of the standard GA, compared to the comparison with population size 10 (1.19/0.84 = 1.4, vs. 3.66/1.84 = 2.0).

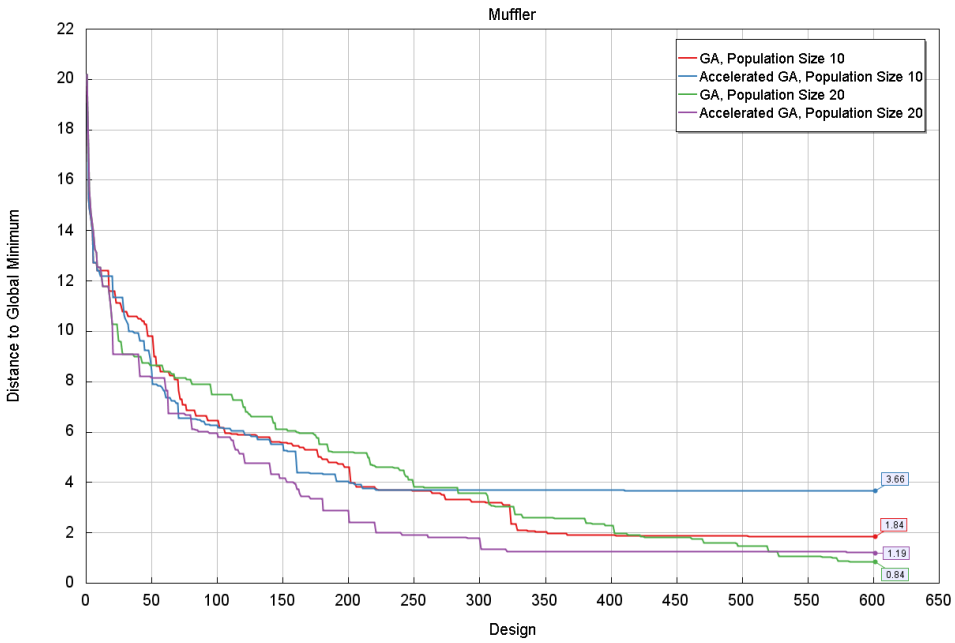


Figure 12: Median progress for the Muffler model using the two algorithms and two population sizes.

7. Conclusions

This paper has shown how the GT-SUITE genetic algorithm's population size and number of generations affect the algorithm's performance, which is measured using median progress among multiple trials and the variance among those trials. The population size is an important setting, where lower values allow faster learning and progress. However, lower values introduce the risk of the population having inadequate diversity, thereby causing the algorithm to converge to a local optimum and not toward the global one. The GT-SUITE genetic algorithm's exploitation process has been improved by incorporating metamodeling using the optimization data. Fast intermediate optimizations between each generation provide an estimate for where the real optimum might exist. The resulting Accelerated GA provides better performance than the standard genetic algorithm for almost all the mathematical and physics-based test models. As a result, the modeler can use larger population sizes and fewer generations to increase diversity without substantially increasing the total number of design iterations. The Accelerated GA should provide the modeler more confidence in finding better optimization solutions in fewer total design iterations.

Gamma Technologies is committed to continued research, testing, and improvement of its optimization and data analysis capabilities to provide its users with increased confidence in using these tools for their own projects. For any questions, support, or further discussion, please reach out to us at support@gtisoft.com.

References

[1] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," in *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577-601, Aug. 2014.

[2] Hadka, David (2015). MOEA Framework User Guide. Version 2.6, available at http://www.moeaframework.org/

[3] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, April 2002.

[4] Ye, Q. (2013). Kernel-based Methods for Stochastic Partial Differential Equations.

[5] Wood, M., & Eames, M. (2017). Using Kriging regression to improve the stability and diversity in NSGA-II.

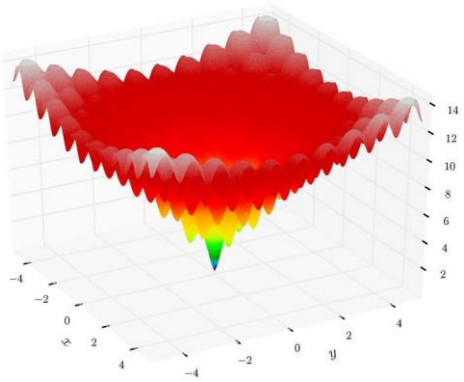
Appendix A

Descriptions of the Optimization Test Models. *R* refers to the response (function output). Graphics are taken from https://en.wikipedia.org/wiki/Test_functions_for_optimization and https://www.sfu.ca/~ssurjano/schwef.html.

Ackley

A 2-variable mathematical function. The global minimum is zero, which occurs when the two factors are zero. The factor ranges were [-5.12, 5.12].

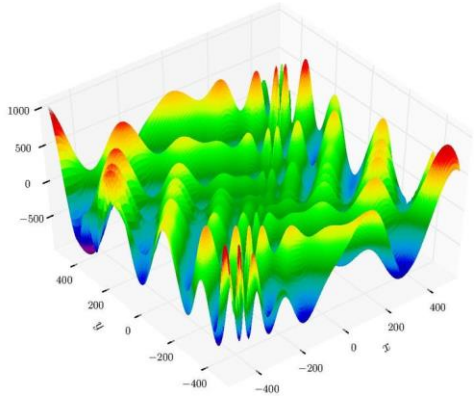
$$R = -20exp\left[-0.2\sqrt{0.5(x_1^2 + x_2^2)}\right] - exp[0.5(cos2\pi x_1 + cos2\pi x_2)] + e + 20$$



Eggholder

A 2-variable mathematical function. The global minimum is -959.6407, which occurs when x_1 is 512.0 and x_2 is 404.2319. The factor ranges were [-512, 512].

$$R = -(x_2 + 47)sin\sqrt{\left|\frac{x_1}{2} + x_2 + 47\right|} - x_1 sin\sqrt{|x_1 - x_2 - 47|}$$



Quadratic

An n-variable mathematical function. With a global minimum of zero, which occurs when all the factors are zero.

$$R = \sum_{i=1}^{n} x_i^2$$

This function was tested with four factors and was modified so that the optimum does not exist at the exact center of the search space. The factor ranges were [-10, 10].

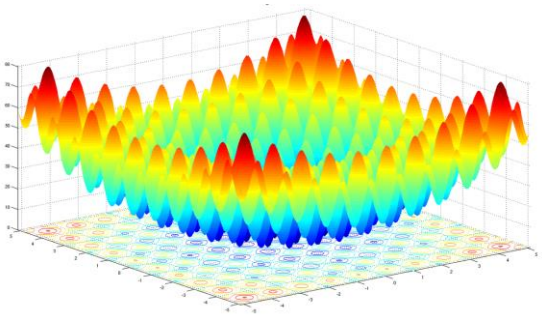
$$R = (x_1 - 3.3)^2 + (x_1 + 7.4)^2 + (x_1 - 8.85)^2 + (x_1 + 2.7)^2$$

Rastrigin

An n-variable mathematical function with a global minimum response value of 0, which occurs when the factors values are zero. The variable ranges were [-5.12, 5.12].

$$R = 10n + \sum_{i=1}^{n} \left[x_i^2 - Acos(2\pi x_i)\right]$$

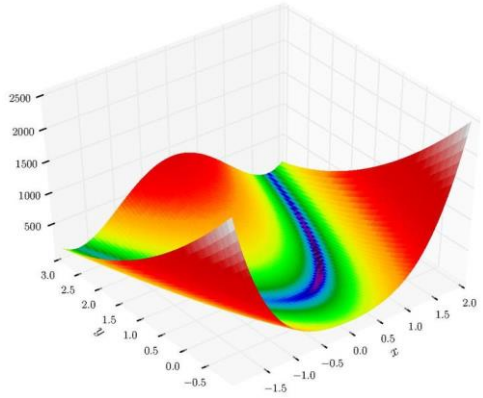
This function was modified by applying a shift to each factor so that the global minimum does not exist at the exact center of the search space.



Rosenbrock

An n-variable mathematical function. The global minimum is zero, which occurs when all factors are equal to 1.0. The factor ranges were [-5, 5].

$$R = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i)^2 + (1 - x_i)^2]$$



Schwefel

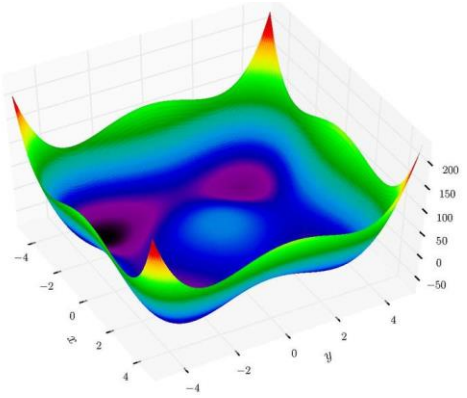
An n-variable mathematical function. The global minimum is zero, which occurs when all the factor values are 420.9687. The factor ranges were [-500, 500]

$$R = 418.9829n - \sum_{i=1}^{n} x_i sin\left(\sqrt{|x_i|}\right)$$

ST (Styblinski-Tang)

An n-variable mathematical function. The global minimum is -39.16617*n, which occurs when all the factor values are 1.0. The factor ranges were [-5, 5].

$$R = \frac{\sum_{i=1}^{n}\left(x_i^4 - 16x_i^2 + 5x_i\right)}{2}$$



DIPulse-A

A predictive diesel combustion calibration that uses the four main DIPulse calibration parameters (entrainment, diffusion, ignition delay, and premixed multipliers) as optimization factors. The response is the calculated burn rate error, and the global minimum is 0.0028719. When evaluating the optimizer performance with respect to the distance to the global minimum, generally values less than 1E-5 (distance to global minimum) are insignificant.

DIPulse-B

A DIPulse calibration that is similar to DIPulse-A, but with measured data from a different engine. Aside from the four main DIPulse calibration parameters, this model uses three addition multipliers related to diffusion, resulting in 7 total optimization factors. The response is the calculated burn rate error, and the global minimum is 0.003. When evaluating the optimizer performance with respect to the distance to the global minimum, generally values less than 1E-5 (distance to global minimum) are insignificant.

DOC

A diesel oxidation catalyst model that solves a reaction-diffusion system for a mechanism that contains six reactions. Reaction rate constants for five of the reactions are parameterized and used as optimization factors. Five pre-exponent multipliers and five activation energies are optimized, for a total of 10 optimization factors. The catalyst model runs through four different light-off simulations, each of which represents a different gas composition. The predicted transient outlet species concentrations are compared to a set of "measured" transient profiles, and a root-mean-squared error is calculated between predicted and measured transient profiles. The objective of the optimization is to minimize the root-mean-square error. The "measured" transient profiles were generated from a simulation, so there exists a single set of exact reaction rate constants that generates zero error, therefore the global minimum is zero. The population size was 50.

Engine

A four-cylinder spark-ignited engine model that runs at six steady-state engine speeds that range from 800 to 5500 RPM. The nine optimization factors include intake valve timings, exhaust valve timings, air-fuel ratios, and spark timings. These parameters vary linearly with engine speed, so their low-speed and high-speed values are the optimization factors. The output variables are IMEP, exhaust temperature, and volumetric efficiency. These output variables (at each of the six speeds) were predicted at a random combination of the factor values and used as target values. As a result, the global minimum is zero. The population size was 50.

Engine Knock

A four-cylinder spark-ignited engine model that runs at five steady-state engine speeds that range from 800 to 4000 RPM. The optimization factors are EGR fraction, intake valve timing, exhaust valve timing, spark timing, and an intake runner pipe length. The four factors that exclude runner length are allowed to vary by engine speed, thereby creating 20 optimization factors. The runner length is required to be a single value for all engine speeds, and as a result, there are 21 total optimization factors. The optimization objective is to maximize brake torque while avoiding knock, which is an optimization constraint. If knock is predicted, the brake torque response is replaced with a very small value. The optimization was artificially transformed to a minimization problem. The global minimum is -182.66. The population size was 50.

GPF

A gasoline particulate filter that simulates soot loading with an inlet particle size distribution. The model predicts the transient pressure drop and number of particles escaping the filter, and the response is the root-mean-square error between these transient results and measured profiles. The global minimum is 2.197. The population size was 50. This model was unintentionally run for 100 generations.

Muffler

A two-chamber muffler model that is discretized into many smaller sub-volumes and run through a transmission loss simulation. The muffler model can be found in the GT examples directory: Examples > Engine_1D_Gas_Exchange_Combustion > Gasoline > SI_4cyl_Basic. The three optimization factors are

baffle position, baffle hole diameter, and outlet pipe diameter. A curve of transmission loss vs. frequency was generated using a random combination of these three factors values. The objective of the optimization is to minimize the error between the predicted transmission loss curve and the generated one. The global minimum is zero, and the population size is 10.

PNA

A passive NOx absorber catalyst model that includes five reaction rate constants used as optimization factors. The global minimum is 14.5354, and the population size is 30.

SCR

A selective catalytic reduction catalyst model that includes two sites for ammonia storage. The model simulates ammonia adsorption and desorption at two different initial temperatures, so only four reactions are active (absorption and desorption on the two sites). There are five optimization factors. The global minimum is 137.88, and the population size is 20.

SITurb

A predictive spark-ignited combustion calibration that uses the four main SITurb calibration parameters (flame kernel growth multiplier, turbulent flame speed multiplier, taylor length scale multiplier, and dilution exponent multiplier) as factors. The response is the calculated burn rate error, and the global minimum is .00225. When evaluating the optimizer performance with respect to the distance to the global minimum, generally values less than 1E-5 (distance to global minimum) are insignificant. The population size is 30.