Data Farming the Parameters of Simulation-Optimization Solvers

SARA SHASHAANI, Industrial and Systems Engineering, North Carolina State University at Raleigh, Raleigh, United States
DAVID ECKMAN, Texas A&M University College Station, College Station, United States
SUSAN SANCHEZ, Operations Research, Naval Postgraduate School, Monterey, United States

The performance of a simulation-optimization algorithm, a.k.a. a solver, depends on its parameter settings. Much of the research to date has focused on how a solver's parameters affect its convergence and other asymptotic behavior. While these results are important for providing a theoretical understanding of a solver, they can be of limited utility to a user who must set up and run the solver on a particular problem. When running a solver in practice, good finite-time performance is paramount. In this article, we explore the relationship between a solver's parameter settings and its finite-time performance by adopting a data farming approach. The approach involves conducting and analyzing the outputs of a designed experiment wherein the factors are the solver's parameters and the responses are assorted performance metrics measuring the solver's speed and solution quality over time. We demonstrate this approach with a study of the ASTRO-DF solver when solving a stochastic activity network problem and an inventory control problem. Through these examples, we show that how some of the solver's parameters are set greatly affects its ability to achieve rapid, reliable progress and gain insights into the solver's inner workings. We discuss the implications of using this framework for tuning solver parameters, as well as for addressing related questions of interest to solver specialists and generalists.

CCS Concepts: • **Computing methodologies → Simulation evaluation**; *Modeling and simulation*; • **General and reference → Experimentation;**

Additional Key Words and Phrases: Data farming, simulation optimization, design of experiments, parameter tuning, ASTRO-DF

ACM Reference Format:
Sara Shashaani, David Eckman, and Susan Sanchez. 2024. Data Farming the Parameters of Simulation-Optimization Solvers. *ACM Trans. Model. Comput. Simul.* 34, 4, Article 24 (August 2024), 29 pages. https://doi.org/10.1145/3680282

1 Introduction

Many *stochastic-optimization algorithms* (a.k.a. *solvers*) have a multitude of settings that influence how they operate, and in turn, how rapidly and reliably they make progress. For example, a basic

This work was partially supported by National Science Foundation Grants CMMI-2226347 and CMMI-2206972.
Authors' Contact Information: Sara Shashaani, Industrial and Systems Engineering, North Carolina State University at Raleigh, Raleigh, North Carolina, United States; e-mail: sshasha2@ncsu.edu; David Eckman, Texas A&M University College Station, College Station, Texas, United States; e-mail: eckman@tamu.edu; Susan Sanchez, Operations Research, Naval Postgraduate School, Monterey, California, United States; e-mail: smsanche@nps.edu.

ACM 1558-1195/2024/08-ART24
https://doi.org/10.1145/3680282

version of the well-known **stochastic gradient descent (SGD)** algorithm may have several user-specified settings, such as the step size, the sample size, and the type of projection operator [37]. A more sophisticated version may allow users to choose a line-search routine [36] or parameters for shifting and scaling the underlying problem [5]. We refer to such user-specified settings or parameters as *solver factors*, which may be real-valued, integer-ordered, or categorical.

In our opinion, the question of how best to investigate, characterize, and set solver factors has not received as much attention as it deserves. The consideration it has received often concerns the solver's asymptotic performance, e.g., local or global convergence and associated rates, accompanied by supporting theoretical results. For the SGD algorithm, for instance, having the sequence of step sizes shrink to zero sufficiently slowly can help to ensure convergence to a local optimal solution. These results are of theoretical value, but they do not always translate to practical recommendations for everyday users interested in achieving good finite-time performance. In particular, tuning parameters based on a problem's properties, e.g., its Lipschitz constant, dimension, or signal-to-noise ratio, remains a challenge. Moreover, factor settings that lead to desirable asymptotic performance do not necessarily yield good finite-time performance. In practice, if a user is likely to run a solver only once on their problem of interest, good finite-time performance—namely, the ability to quickly and consistently find better solutions—is paramount. In research, identifying the most influential factors for achieving good performance for all problems (if possible), or for specific classes of problems (e.g., those with convex or nonsmooth objective functions, or those of low or high dimension), can provide a structured and holistic approach for identifying default settings and developing guidance for users, so paving the way for a new generation of robust and reliable solvers.

1.1 Related Work

Past research on setting solver factors has predominantly focused on search techniques for tuning hyperparameters of machine-learning algorithms. Unlike in the simulation-optimization context we consider in this article, these algorithms typically do not encounter any randomness in the objective function evaluations [49, 50] and therefore do not exhibit variable performance from run to run. These techniques include, but are not limited to: Bayesian optimization algorithms [1, 31], which are limited by their dependence on certain assumptions and computational cost when the space of hyperparameters grows; successive halving and bandit methods [27], which are limited by their sensitivity to random sampling; and combinations of the two [13], which can be limited by early stopping and the aforementioned scalability challenges of Bayesian optimization. Random search [2] and evolutionary algorithms [35] are also often used for this purpose, but have limitations in computation, parametric sensitivities, and a lack of convergence guarantees.

Many other studies have addressed the tuning of continuous-valued hyperparameters by sequentially generating and evaluating candidate configurations of hyperparameters of an optimization algorithm. Iterative procedures have been devised to generate a new set of candidates via **design of experiments (DOE)** at the rudimentary level of full factorial, Latin hypercube, or random sampling designs [15], probability models [33], neighborhood search [24], or optimization algorithms including response surface methods and numerical gradient-based algorithms or heuristics [21, 22]. The solver's performance under a given configuration is assessed via fixed evaluation [51], screening [3], or an adaptive number of runs [23]. Many of these methods aim to find a promising region in the parameter space at minimum cost. For a recent survey, see [20].

Other gaps exist in the literature. A persistent challenge when searching for good hyperparameters for solving a specific problem has been the development of a framework that easily accommodates categorical parameters and a large hyperparameter space. Moreover, when choosing the best hyperparameter configuration for a solver for stochastic optimization, the variability

(risk) of the solver's performance under each configuration has rarely been incorporated [40]. In addition, reporting a configuration as optimal for a solver without knowing why it is optimal is of limited utility. Such an approach does not lend flexibility to see, for example, synergistic or redundant interaction effects among hyperparameters or between hyperparameters and problem features, e.g., the problem dimension. Consequently, this kind of reporting does not inform the user on how the best configuration might change if the problem being solved were to change in size, features, or certain constant factors.

When algorithms for hyperparameter tuning are not employed, experimentation with solver factors is often *ad hoc*. In the extreme case, experiments are run with solver factors set to what appear to readers to be arbitrary values. These values may be those that produced the best empirical performance on the set of test problems, perhaps chosen to make a newly proposed solver look good relative to its competitors. Even when experiments have been run with a variety of factor settings, it can be difficult to extract practical guidance on how to set factors for a new problem. This issue is compounded by the reality that a solver's "out-of-the-box" settings in software implementations may not be good for all problems. Moreover, these default settings may not even be the product of extensive experimentation.

1.2 Our Contributions

We seek to address the limited understanding users and developers may have about how a solver's factors affect its behavior and finite-time performance. We approach the problem from a data-farming perspective wherein solver factors are systematically varied according to a DOE. Our proposed approach entails running extensive experiments with different versions of a solver on a given problem or a small class of problems. Our use of DOE differs from past research on data farming, which generally concerns varying parameters of a simulation model to understand how changes to its inputs affect its performance measures. We instead vary parameters of the solver, and our response functions are performance metrics for simulation-optimization algorithms, such as the time at which the solver attains some specified improvement in the objective. Our approach can be extended to a range of algorithms that touch upon many domains. For example, algorithms for surrogate optimization [19], derivative-free optimization [26], stochastic programming [4], and Bayesian optimization [14] can all exhibit variable run-to-run performance and could benefit from more informed setting of solver factors. Other approaches that have been observed to be sensitive to the setting of hyperparameters are **simulation-based inference (SBI)** [28] and some types of metamodeling [12]. To the best of our knowledge, DOE and data farming have not been applied in the context of studying solver performance when incorporating the solver's variability, but we demonstrate that they can be powerful methods for gaining a deeper understanding of a solver. The advantage of DOE for experimenting with solver factors is that it can provide more than just an optimal configuration.

Different types of researchers or analysts can have different end goals when using this approach. Consequently, they might benefit in different ways, either directly or indirectly. We consider three groups. First, those interested in general solver performance could use this approach to assist in setting appropriate default values for a particular solver's factors during development, provide guidance to solver users about when and how they might seek to improve on these default values, or perhaps choose to remove some as factors and re-specify them as functions of other factors. We refer to these as *solver generalists* or simply *generalists* throughout the remainder of this article. We anticipate that providing insights into the tradeoffs, risks, and sensitivities of a solver's inner workings will assist generalists in advancing the state of the art in simulation-optimization methodology. Second, those faced with the need to solve a particular class of problems on a regular basis might seek to tune the solver's factors in ways that are beneficial for that class. We will refer

to these sophisticated users as *solver specialists* or simply *specialists*. Specialists directly benefit from this approach, because even small improvements in the quality and timeliness of solutions to individual problems may scale to massive overall improvements when hundreds or thousands of problems are solved. Finally, occasional solver users also stand to benefit, albeit indirectly. They may care about the solver parameters not at all, focusing solely on finding a high-quality solution to a particular problem by making a limited number of runs of one or more solvers. We will refer to these as *occasional focused users* or simply *focused users*. Focused users may care little about differences in finite-time performance with different parameter configurations (or different solvers) as long as a good solution is found in a reasonable time; if they have a month to solve a problem, solving it in one hour, one day, or one week may be equally valuable. However, in the short term, focused users might indirectly benefit from using a particular solver whose default parameter settings have been chosen wisely. Similarly, if they decide to make more than one run of the solver, perhaps because they are not sure they have a good solution from the first run, focused users would benefit from having reasonable guidance on how to modify the solver's parameters on subsequent runs. In the longer term, focused users will benefit as better solvers are developed. As a side note, much of the published discussion on tuning hyperparameters apparently focuses on benchmarking studies for this type of user [11]; we believe the current wave of increased reliance on analytics and AI will draw many more specialist users into the fold. While focused users could use a data farming approach, they are less likely to do so unless the methodology is embedded into the solver itself. Consequently, generalists and specialists are our primary audience in this article.

The rest of this article is organized as follows: In Section 2, we introduce the canonical simulation-optimization problem and a set of metrics for evaluating a solver's finite-time performance. We then set up a data-farming framework for studying solver performance in Section 3. In Section 4, we present a list of "Top Ten" questions and associated statistical techniques that can guide an analysis of the experimental results. In Section 5, we apply these techniques to study the ASTRO-DF algorithm and demonstrate that how the solver's factors are set matters, in a measurable way, and illustrate some of the types of attainable insights that extend beyond optimum seeking or parameter tuning. We conclude in Section 6 with a discussion of extensions to the data-farming framework and its implementation in the SimOpt testbed [10].

2 Simulation Optimization and Solver Performance Metrics

A **simulation-optimization (SO)** problem is an optimization problem where the objective function and/or constraints are evaluated (with noise) through a stochastic simulation. A wide variety of problems can be formulated as SO problems. We consider the prototypical SO problem:

$$\min_{x} \; f(x) \equiv \mathbb{E} f(x, \xi) \;\text{ subject to } x \in \mathcal{D}, \tag{1}$$

where x is a vector of decision variables that lies in a domain $\mathcal{D}$. We consider the setting in which $\mathcal{D}$ is either unconstrained, i.e., $\mathcal{D}$ solely restricts components of x to be integer-valued or continuous, or is described by only deterministic constraints; we do not consider stochastically constrained domains in this article. The term ξ represents the collection of random variables generated by a simulation model in the course of simulating a single replication. The distribution of ξ is allowed to depend on x, thus the expectation in Problem (1) is over realizations of ξ for a given x. The function $f(x, \cdot)$ denotes the logic used to obtain an estimate of the objective function $f(x)$ for a given feasible solution x.

2.1 Measuring Solver Performance

An SO solver attempts to solve Problem (1) by estimating the objective function values of different feasible solutions. For example, at a given solution x, a solver may take n_x simulation replications

and estimate $f(x)$ by the sample average of the outputs $f(x, \xi_1), f(x, \xi_2), \ldots, f(x, \xi_{n_x})$. Depending on the problem and available computational resources, it may be impossible to evaluate all solutions in $\mathcal{D}$; instead the solver may evaluate only a subset of feasible solutions. A solver strives to quickly find feasible solutions with near-optimal objective function values. Although SO algorithms are diverse, some efforts have been made to develop widely applicable metrics for evaluating their performance. For example, performance profiles as well as data profiles [32] track the solvability (percentage of problems solved by a solver—a.k.a., success ratio) given a fixed time (computation budget), and overall efficiency metrics [48] track the trade-off between computation time and solvability. We adopt and adapt many of the metrics introduced in [8] and provide only a summary description here to convey the essential ideas.

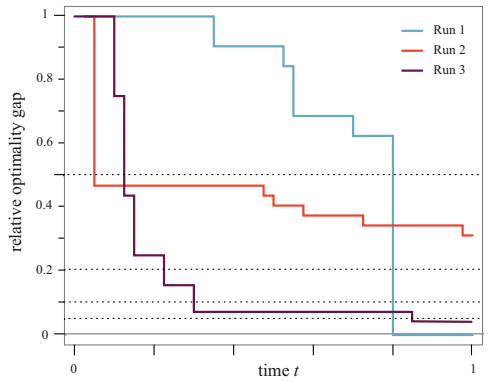
When running a particular SO solver on a given problem, we fix a budget T—measured in simulation replications—and observe the sequence of solutions recommended by the solver as it expends the budget. This sequence of recommended solutions can be conveniently viewed as a continuous-time stochastic process $\{X(t)\colon 0 \leq t \leq 1\}$ where $X(t)$ is the solution recommended when a fraction t of the budget T has been expended. The objective function values of these solutions also form a stochastic process $\{f(X(t))\colon 0 \leq t \leq 1\}$. Plotting $f(X(t))$, or an estimate thereof, as a function of t is common in the optimization literature. To produce metrics that are independent of the scaling of the objective values and of the direction of optimization (minimization vs maximization), one can standardize $f(X(t))$ using the objective function values of some (typically poor) initial solution x_0 and a reference optimal solution x^*. To be more precise, consider

$$v(t) = \frac{f(X(t)) - f(x^*)}{f(x_0) - f(x^*)},$$

which represents the relative optimality gap of the solution recommended at t, relative to the initial optimality gap. A realization of the stochastic process $\{v(t)\colon 0 \leq t \leq 1\}$ is called a *progress curve* of the solver. A plot of $v(t)$ as a function of t shows a piecewise-constant function that is typically, though not necessarily, nonincreasing. Because the objective function f cannot be evaluated exactly for SO problems, one would more commonly work with an estimated progress curve obtained by estimating the objective f using replications of the simulation model at x_0, x^*, and $X(t)$. Using the outputs of the replications taken by the solver when visiting a solution $X(t)$ is known to produce an optimization bias [29]. Thus, it is advisable to take fresh independent replications (referred to as *postreplications*) in a post-processing stage. The choices for the number of postreplications for each $X(t)$ and the number of postreplications for the reference solutions x_0 and x^* are user-specified and could be chosen to be relatively large to avoid incorrectly ordering solutions by chance. See Section 8 of [8] for a detailed discussion and a theoretical investigation of these choices.

A progress curve contains a great deal of information about the performance of a solver on a single run. For example, Figure 1(a) shows progress curves for three different runs of a fictitious solver. In Run 1, the solver takes a long time to begin improving on its initial solution, but finds the optimal solution after 80% of the budget has been consumed. By contrast, the curve for Run 2 shows the solver making great progress initially, followed by a long period where no progress is achieved before gradually reducing the optimality gap. In Run 3, the relative optimality gap drops fairly early (though not as soon as in Run 2), but the solver later has a more difficult time improving on its solution.

We will find it helpful to summarize the information contained in a progress curve with a few scalar metrics, which can then be treated as responses within our data-farming framework. We highlight four functionals of the progress curve that describe important aspects of a solver's performance:



(a) Progress curves for three runs. Dashed lines indicate thresholds of 0.05, 0.5, 0.2, and 0.1 for the relative optimality gap.

Solver metric	Run 1	Run 2	Run 3
Final relative optimality gap $v(1)$	**0.00**	0.05	0.31
Area under progress curve A	0.69	0.43	**0.21**
Time τ at which curve drops below threshold			
$\tau(0.5)$	0.80	**0.05**	0.15
$\tau(0.2)$	0.80	n/a	**0.23**
$\tau(0.1)$	0.80	n/a	**0.30**
$\tau(0.05)$	**0.80**	n/a	0.85
Curve drops below threshold			
$y(0.5)$	**1**	**1**	**1**
$y(0.2)$	**1**	0	**1**
$y(0.1)$	**1**	0	**1**
$y(0.05)$	**1**	0	**1**

(b) Solver performance metrics for the three runs. The bests of each measure are indicated in bold.

Fig. 1. Progress curves and performance metrics for three runs of a solver.

— The *final relative optimality gap*, $v(1)$, indicates what fraction of the initial optimality gap is remaining after the solver has exhausted its budget.
— The *area under the progress curve*, $A = \int_0^1 v(t)\,dt$, is the time-average relative optimality gap associated with the solutions recommended by the solver over time.
— The *α-solve time*, or time at which the relative optimality gap first drops below some threshold $\alpha \in [0, 1]$, $\tau(\alpha) = \inf\{t \in [0, 1] : v(t) \leq \alpha\}$, measures the speed with which the solver achieves a specified degree of improvement.
— The *α-solvability*, or whether or not the relative optimality gap drops below some threshold $\alpha \in [0, 1]$ before termination, $y(\alpha) = \mathbb{I}\{\tau(\alpha) \leq 1\}$, measures the ability of the solver to find a near-optimal solution given its budget.

Figure 1(b) displays the solver performance metrics associated with the three progress curves from Figure 1(a). Each of the runs is associated with the best performance in at least one metric, but none of the runs dominate in all.

The behavior of an SO solver varies from run to run due to random error when estimating the objective function and (optional) intrinsic randomness, e.g., picking a random search direction. It is therefore imperative to conduct multiple runs, hereafter referred to as *macroreplications*, of a solver on a problem to understand its variable performance. In light of this variability, the scalar metrics listed above can be regarded as random variables, and properties of their distribution, e.g, their mean, median, variance (when finite), high-probability quantiles or **VaR (Value at Risk)**, and **CVaR (Conditional Value at Risk)** [18] serve as useful summary statistics.

2.2 Solver Factors

SO solvers typically feature many parameters that affect how the solver evaluates solutions and seeks better ones. In a nod to the conventional terminology used in DOE, we call such parameters *solver factors*. Solver factors come in many forms and can control a myriad of aspects of solver behavior. For example, solver factors can relate to an exploration-exploitation tradeoff within an algorithm, such as the choice of the population size in a genetic algorithm, the step length in a gradient-descent algorithm, or the criteria for choosing a model-minimizing point within a trust-region algorithm. As illustrated in these examples, solver factors can be integer-ordered, real-valued, or categorical. These factors influence not only the solver's behavior, but also the solver's average and variable performance from run to run, i.e., its ability to consistently find high-quality solutions.

As a practical remark, to carry out the experiments we discuss in the next section, solvers should be coded in a way that allows their parameters to be varied externally. In this way, an experiment involving multiple versions of the same solver can be run without needing to manually change hard-coded values. Many code implementations of SO solvers are not structured in this way, since only one version of the solver is typically run when solving a problem in practice. One exception is the SimOpt library [10], an open-source testbed of SO problems and solvers, wherein solvers are coded to support flexible experimentation where any and all solver factors may be altered. The solver performance metrics discussed in Section 2.1 and the data-farming framework we are about to introduce have been implemented in a recent redesign of SimOpt, described in [9], and users can reproduce the experiments in this article or run their own on the library's collection of problems and solvers.

For Solver s, we let $s(v)$ denote a *version* of the solver corresponding to its factors being set as v. As previously mentioned, v may represent a combination of numerical and non-numerical factors. We wish to understand how performance metrics of Solver s, particularly those listed in Section 2.1, change as the combined factor settings, v, change. To study this question, we introduce a data-farming framework that involves testing multiple versions of the solver and analyzing the results.

3 Experiment Design

Studies of new SO algorithms commonly feature experiments that test only a handful of versions of a solver, with these often coming from varying one or two solver factors. *Ad hoc* comparisons like this allow only limited conclusions to be drawn about the impact of solver factor settings on solver performance. Varying individual factors one at a time while holding the other fixed offers in sight into only the marginal effects of each factor, which may be of little informational value because it precludes the identification and estimation of interaction effects and makes it impossible to fit metamodels that capture solver behaviors in the interior of the factor space. Moreover, full-factorial designs, wherein all combinations of setting solver factors at a finite number of levels are evaluated, can be too expensive to run, even for modest numbers of factors and levels.

DOE affords a more systematic comparison of the performances of different versions of a solver with a more efficient (i.e., smaller) design. For concreteness, consider a set of r versions of Solver s, denoted by $\mathcal{S} = \{s(v_1), s(v_2), \ldots, s(v_r)\}$ where the versions differ only in the settings of the k solver factors under investigation. In the DOE terminology, the set $\mathcal{V} = \{v_1, v_2, \ldots, v_r\}$ is called the *design*, and each combination of factor settings, v_j for $j = 1, 2, \ldots, r$, is called a *design point*. The design $\mathcal{V}$ should possess certain properties suited to the user's needs, such as filling the space of factors or allowing for the study of interactions among factors. There are many good designs, but some are more suitable for physical experiments or deterministic computer experiments than for stochastic simulation experiments. A few designs we recommend for investigating continuous-valued solver factors are **nearly orthogonal Latin hypercubes (NOLHs)** [7], **resolution V frequency based designs (R5FBDs)** [39], or **resolution V central composite designs (R5CCDs)** [41]. These all explore factors at multiple levels, and provide flexibility for fitting different types of metamodels or using different visualization techniques during analysis. Orthogonality or near-orthogonality is desirable for separately estimating the effects of different factors. The R5FBD and R5CCD are perfectly orthogonal and permit all main effects, quadratic effects, and two-way interaction effects to be fit simultaneously. If some factors are discrete with relatively few levels, or categorical factors, then **nearly orthogonal-and-balanced (NOAB or NOB designs)** [47] can be used; here, "nearly balanced" refers to sampling each level of a discrete-valued or categorical factor approximately the same number of times. The space-filling

behavior for most of these designs can be improved by reassigning factors to columns of the design matrix, a process called *stacking*. Finally, *crossing* two designs $\mathcal{V}_1$ (k_1 factors, r_1 design points) and $\mathcal{V}_2$ (k_2 factors, r_2 design points) is a straightforward way to construct designs capable of exploring k_1+k_2 factors in $r_1 \times r_2$ design points; these can be useful, even though they are not the most efficient.

Those unfamiliar with the DOE field may not realize how extremely efficient these designs can be. For instance, consider a solver with 20 factors. If one were to use a brute-force method to explore these factors simultaneously, each at 2 levels, that would require 2^{20} or nearly 1.05M design points; exploring these at 10 levels each would require 10^{20} or 14 orders of magnitude more. Conversely, the 20 factors could be studied at 129 levels in 129 design points (NOLH), at 1673 levels in 1673 design points (R5FBD), at 5 levels in 553 design points (rotatable R5CCD), or varying numbers of levels in 512 design points (NOAB). See [43] for more information about DOE for simulation experiments, or see [38] to download design generator software.

DOE plays a central role in the area of data farming [40]. In data farming, a user studying a simulation model first "grows" a dataset by running the model with different combinations of inputs (i.e., factors of the model) according to a design of experiments and then analyzes the results. Our application of data farming differs from the conventional approach in that our responses of interest are not the performance indicators of the simulation model but rather metrics of how well a solver solves a given SO problem, such as those mentioned in Section 2.1.

Our experiment design entails running multiple macroreplications of each version of the solver (corresponding to each design point) on a fixed SO problem. The same problem instance is solved at all design points, hence, the progress curves and all associated metrics are standardized using the same reference solutions x_0 and x^*. Specifically, all macroreplications of all solver versions start at the same initial solution x_0, and x^* is taken to be the estimated best solution recommended by any solver on any macroreplication. We perform a common number of macroreplications at each design point, though the data-farming framework permits the sample sizes to vary across design points. For instance, if the observed responses (solver performance metrics) at a given design point were highly variable, we may choose to take more macroreplications.

When taking a common number of macroreplications at all design points, **common random numbers (CRN)** can be used across design points to sharpen the comparisons. In our setup, CRN would ideally mean that the different versions of the solver would see the same sequences of objective function estimates if simulating the same solutions and would all use the same sequence of random inputs for their internal purposes. The former of these is readily achieved if the solver additionally using CRN across solutions. The experiments in this paper were conducted in SimOpt [10] and use CRN in all these ways by default. Our intention in using CRN across design points is to better estimate the *ordering* of performance metrics when comparing multiple versions of the solver.

4 Motivating Questions for Analysis

The experiment introduced in Section 3 produces a large amount of data, consisting of multiple solver performance metrics tracked on each macroreplication of each solver version tested. It remains to transform this data into insights via a purposeful analysis. In this section, we outline pertinent questions to the study of solver performance and the statistical techniques best suited for answering them. The intent is to showcase the range of analyses that can be carried out on the data produced by the experiment. In Section 5, we present an empirical study, going into much more depth and discussion on the methods and potential insights.

Table 1 lists the "Top Ten" questions that have been proposed as guides to the analysis of data-farming experiments, along with a partial summary of statistical tools and visualization techniques

Table 1. "Top Ten" Questions to Ask During the Analysis of Results from a Solver Performance Experiment

Question	Some techniques
1. What was the spread of solver performance metrics over the entire experiment?	summary statistics, histograms, box plots
2. What run-to-run variation (spread, shape, and central tendency) was observed for the solver performance metrics of interest? Did these vary across design points?	
3. Are there any outliers or solver factor configurations that lead to unusual behavior?	box plots, scatter plots, metamodel-specific diagnostics (e.g., Cook's distance for regression metamodels)
4. How are the solver performance metrics correlated? Do they involve tradeoffs or behave serendipitously?	correlation matrices, scatterplot matrices
5. Which solver factors are most influential?	stepwise regression, partition trees, other metamodels, interaction profiles, interactive graphs
6. Are there any important interactions?	
7. What are the interesting regions and threshold values in the solver factor space?	
8. Are any of the results counterintuitive?	trace results to their causes as part of verification and validation
9. Which configurations of solver factors are most robust, leading to consistently good performance for a solver metric of interest?	contour plots, metamodels of robustness as quantified by loss functions over solver performance metrics, risk-adjusted solver performance metrics
10. Are there any configurations that perform well w.r.t. multiple solver performance metrics?	parallel plots, heatmaps, defining constraints, Pareto optimal curves, weighted loss functions

that can be used to answer these questions [34]. These are not intended to be exhaustive, but are adapted from similar questions that have helped direct the analysis for hundreds of data-farming experiments that investigate the relationships between *simulation model factors* and *simulation model responses* for specific applications. Our "Top Ten" questions shift the focus to exploring the relationship between *solver factors* and *solver performance metrics*.

In Section 1.2, we stated that different types of researchers and analysts can benefit in different ways from such a study. Their overarching goals in answering these questions may differ as well. We now elaborate further.

Generalists (i.e., those interested in general solver performance) could use such a study to assist in setting appropriate default values for a particular solver's factors during development, provide guidance to solver users about when and how they might seek to improve on these default values, or perhaps choose to remove some as factors and re-specify them as functions of other factors. Researchers developing or implementing new solvers can be thought of as generalists. Generalists may also be interested in experimenting with a group of solvers across multiple problems to gain insight into which problems can be characterized as "easy" or "hard" for those solvers; this kind of meta-experiment is beyond the scope of this article, but is a worthwhile direction for future research.

Specialists faced with the need to solve a particular class of problems on a regular basis might seek to tune the solver's factors in ways that achieve good finite-time performance for that class. For example, a retail warehouse might be faced with setting inventory policies for thousands of items where the policies are revisited over time as demand patterns evolve, costs change, or new products are introduced. Specialists would benefit from including problem-specific factors (e.g., parameters of the demand distribution input models, holding costs, etc.) in their experiment, as well as solver factors. Such a study could either determine that certain solver factor settings yield good solver performance for a wide variety of potential futures, or provide guidance on how to

select good solver factor settings based on the problem's characteristics. Either way, even small improvements in the quality and timeliness of the solutions may lead to substantive reductions in the overall cost when the specialist is called on to solve thousands of problems. The upfront cost and time required to run the data farming experiment may be quickly recouped by the benefits of improved performance if, say, the specialist can solve problems on a much more rapid basis. Of course, if the problems are relatively simple to solve, they might not need such a study or care about which solver version they chose.

5 A Study of ASTRO-DF

We demonstrate how to leverage data farming to investigate solver factors with an example consisting of one solver and two problems, described in Sections 5.1 and 5.2, respectively. These are drawn from the portfolio of solvers and test problems currently implemented in the SimOpt testbed [10], which continues to expand; the interested reader could conduct a similar study using different solver and problem instances. We set up an experiment in Section 5.3 and then delve into the "Top Ten" questions in Section 5.4. Throughout these sections, we discuss situations where specialists and generalists might pursue different lines of inquiry. Our goal in this section is to provide a general template for those embarking on data farming a solver's parameters. That is, we focus on illustrating the methodology rather than reporting on a definitive study of ASTRO-DF's performance. We discuss further extensions in Section 5.5.

5.1 The ASTRO-DF Solver

ASTRO-DF [16, 17, 45], which stands for Adaptive Sampling Trust-Region Optimization for Derivative-Free problems, is a class of algorithms that use local models fitted on function estimates of an incumbent solution and its neighbors to converge to a stationary point. The neighborhood represents the region within which the local model is credible and its size (specifically, its radius) governs the step size taken to determine the next iterate. ASTRO-DF is proven to converge to a first-order critical point almost surely and reach ϵ-optimality at the rate of $\tilde{O}(\epsilon^{-4})$ under certain assumptions, irrespective of the setting of its factors. This example illustrates the finite-time performance of ASTRO-DF as a function of some of its factors that determine how the trust region is updated; the relevant details of the ASTRO-DF solver are described below.

On an iteration k, ASTRO-DF maintains a trust region of radius Δ_k centered at an incumbent solution X_k, denoted by $B(X_k;\Delta_k) = \{x \in \mathcal{D} : \|x - X_k\| \leq \Delta_k\}$. The algorithm estimates the objective function values at neighboring solutions located on the boundary of $B(X_k;\Delta_k)$ along the coordinate basis. The sample sizes for each neighboring solution are determined by an adaptive sampling rule and are roughly $\tilde{O}(\Delta_k^{-4})$. This ensures that as the trust region contracts, signifying closeness to a stationary point, the accuracy of the estimated objective function values sufficiently increases. ASTRO-DF then fits a quadratic model on these estimated objective function values and performs a quality certification. In the quality-certification step, if the norm of the gradient of the model at the incumbent solution is too small relative to Δ_k, the trust-region radius is reduced and a new quadratic model is constructed. The algorithm then optimizes (minimizes) the certified local model within the trust region $B(X_k;\Delta_k)$, and the resulting solution $\tilde{X}_{k+1}$ serves as a candidate for the next incumbent. The success of the local model is assessed by estimating the objective function at the candidate solution $\tilde{X}_{k+1}$ and comparing the estimated improvement in moving from X_k to $\tilde{X}_{k+1}$ with the improvement predicted by the model. The objective function estimate improvement divided by the model improvement is called the success ratio ρ_k, and ASTRO-DF accepts the candidate solution if $\rho_k \geq \eta_1$ and rejects it otherwise. The trust-region radius Δ_k is updated before the start of the next iteration according to the following rule:

$$\Delta_{k+1} = \begin{cases} \min\{\gamma_1 \Delta_k, \Delta_{\max}\} & \text{if } \rho_k \geq \eta_2 \text{ [very successful iteration – expand trust region],} \\ \Delta_k, & \text{if } \rho_k \in [\eta_1, \eta_2) \text{ [successful iteration – keep trust region unchanged],} \\ \gamma_2 \Delta_k, & \text{otherwise [unsuccessful iteration – contract trust region].} \end{cases} \tag{2}$$

The trust-region radius, and how it changes throughout the search, affects the performance of ASTRO-DF in several ways: it governs the local model construction and certification, and it controls the step size between consecutive iterations. Four user-specified parameters appear in (2): γ_1, the rate of expansion of the trust region after a very successful iteration; γ_2, the rate of contraction of the trust region after an unsuccessful iteration; and η_1 and η_2, the thresholds defining successful and very successful iterations, respectively. We will vary these four factors in a design.

5.2 The Problems

We cannot investigate a solver's behavior without observing its performance. So, we apply the ASTRO-DF algorithm to two problems from the SimOpt library [10]: SSCONT, an (s, S) inventory problem with continuous demand and order quantities, and SAN, a stochastic activity network problem with 9 nodes and 13 arcs. The former is a 2-dimensional problem that is non-convex, whereas the latter is 13-dimensional and convex.

The SSCONT problem features a single product whose inventory level is dictated by an (s, S) inventory policy, meaning that when the inventory level drops below s, an order is placed so that the inventory level can be brought back up to S. The demand in each period is assumed to be exponentially distributed with a mean of 100 units, independent across periods. The order lead time is assumed to be discrete and random, following a Poisson distribution with mean of 6 periods, independent across periods. The objective is to find values of s and S that minimize the expected total cost over 100 periods, which is the sum of total back-order cost ($4 per unit), total holding cost ($1 per unit per period), total fixed cost ($36 per order), and total variable cost ($2 per unit) using a warm-up of 20 periods. The standard formulation of the problem requires that $s < S$, however, in SimOpt, we reparameterize the decision variables to be s and $S - s$ so that the domain is the non-negative orthant $\mathcal{D} = \mathbf{R}^+ \times \mathbf{R}^+$. The optimal solution is unknown.

In the SAN problem, a network of nodes are connected with arcs representing activities in a project; the length of each arc represents the duration of the activity and the direction of each arc reflects the precedence. The objective is to specify parameters of the arc-length distributions to minimize a combination of the length of the longest path and an associated cost. In particular, if $\theta = (\theta_i,\ i = 1, 2, \ldots, 13)$ are the parameters of exponential distributions for arc lengths and $c = (c_i,\ i = 1, 2, \ldots, 13)$ are the associated costs per unit time for arcs 1 to 13, then the objective function is of the form $\mathbb{E}L(\theta) + c(\theta)$, where $L(\theta)$ is the (random) length of the longest path from node `a` to node `i`, i.e., the duration of the project, and $c(\theta) = \sum_{i=1}^{13} c_i/\theta_i$. For this instance of the problem, we let $c_i = 1$ for $i = 1, 2, \ldots, 13$ and use the network structure depicted in Figure 2. The optimal solution to this problem is also unknown.

5.3 The Experiment

Now that we have specified our solver and some problems on which it will be applied, we can set up an experiment to systematically study how the solver factors that control changes to the trust-region size—namely γ_1, γ_2, η_1, and η_2—influence the performance of ASTRO-DF. For each of these factors, we specify a range of values over which we will vary them in a design. For γ_1 in particular, the exploration range is chosen to be $[1.0, 3.0]$ because increasing the trust-region radius (which determines the step size) by more than a factor of 3 seems unreasonable. The other three factors must be between 0 and 1, and additionally we must set $\eta_1 < \eta_2$. In all runs of all variations of the

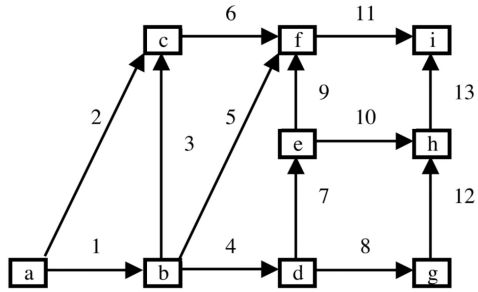


Fig. 2. The stochastic activity network structure used in the experiment.

Table 2. ASTRO-DF Solver Factors and Factor Ranges Used in the Experiment

Factor	Description	Type	Specification	Range
γ_1	expansion rate of Δ_k after a very successful iteration	continuous	$\in (1, \infty)$	$[1.0, 3.0]$
γ_2	contraction rate of Δ_k after an unsuccessful iteration	continuous	$\in (0, 1)$	$[0.1, 0.9]$
η_1	threshold for a successful iteration	continuous	$\in (0, 1)$	$[0.1, 0.5]$
η_2	threshold for a very successful iteration	continuous	$\in (\eta_1, 1)$	$[0.5, 0.9]$

ASTRO-DF solver, we fix the initial solution of SSCONT to be $x_0 = (600, 600)$ with total budget (the maximum number of function evaluations) of $T = 1,000$ and the initial solution of SAN to be $x_0 = (8, 8, \ldots, 8)$ with total budget of $T = 10{,}000$.

For efficiency, our chosen design over solver factors is a nearly orthogonal Latin hypercube (NOLH) design with 3 stacks for a total of $17 + 16 + 16 = 49$ design points. This design was generated using the `datafarming` Ruby gem version 1.4.0 [38] with

```
stack_nolhs.rb -s 3 < astrodf_factor_setting.txt > astrodf_design.txt
```

run in the command line. In the command above, `astrodf_factor_setting.txt` is an input text file specifying the range of settings of the four solver factors, as listed in Table 2, and `astrodf_design.txt` is an output text file (to be created) listing all 49 design points. Each design point corresponds to a version of the ASTRO-DF solver with a specific combination of values for the factors γ_1, γ_2, η_1, and η_2. Note how efficient this is: a full factorial design (gridded design) for 4 factors at 17 levels requires $17^4 = 83521$ design points, while three stacks of our NOLH requires only 49. On a computing cluster with 49 or more cores, this is essentially the same user time as running a single design point.

Many options are possible for the problem factor design. We opt for the simplest in this article: a single categorical factor that specifies the problem name, as shown in the first column of Table 3. In this table, we also list the default settings that, while unchanged in our experiment, might be factors of interest in future experiments. For example, a solver specialist might wish to vary the factors of the SSCONT problem to get a sense of how often it is feasible and useful to re-optimize if costs or demand patterns change, or investigate how long the warm-up period and run lengths should be to achieve satisfactory performance on key solver metrics. A solver generalist might be more interested in exploring how the solver behaves on problems of different dimensions, e.g., by generating a wide variety of stochastic activity networks with different structures and numbers of arcs, or by exploring a larger selection of problems from the SimOpt testbed.

Our overall design is the result of crossing the two individual designs: one for the solver factors, the other for the categorical "Problem" factor, for a total of $49 \times 2 = 98$ design points. For each

Table 3. Problem-Specific Potential Factors and Default Settings for Two Simple Problems

Problem	Factor Description	Type	Specification	Default
SSCONT	demand distribution	categorical		exponential*
	mean demand per period	continuous	> 0	100
	order lead time distribution	categorical		Poisson*
	mean order lead time, in periods	continuous	> 0	6
	back-order cost per unit	continuous	> 0	4
	holding cost per unit	continuous	> 0	1
	fixed cost per order	continuous	> 0	36
	variable cost per unit	continuous	> 0	2
	warm-up before data collection, in periods	integer	≥ 0	20
	number of periods after warm-up in each run	integer	≥ 1	100
SAN	distribution of activity time for arc i	categorical		exponential*
	cost per unit time for arc i	continuous	> 0	1
	number of nodes	integer	≥ 1	9
	graph precedence structure	categorical		see Figure 2

*Not currently farmable in SimOpt.

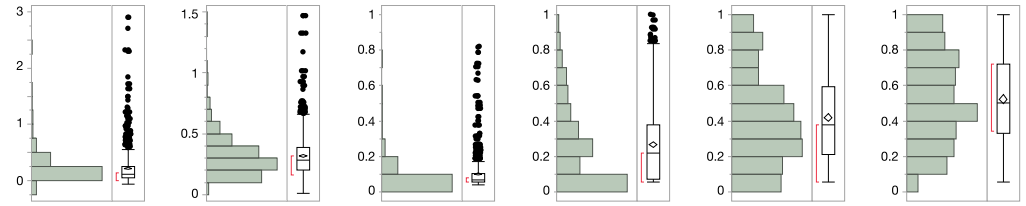
design point, we ran 10 macroreplications of the corresponding version of ASTRO-DF and set the numbers of postreplications as 200 for each recommended solution. On each macroreplication of each design point, we recorded the following 10 responses: the final relative optimality gap $\nu(1)$, the area under the progress curve A, and the α-solve times $\tau(\alpha)$ and α-solvabilities $y(\alpha)$ for $\alpha = 0.05, 0.1, 0.2$, and 0.5. The final relative optimality gaps and α-solvability responses reflect the goodness of the solutions that ASTRO-DF can find given the budget. On the other hand, the area under the progress curves and α-solve times reflect the speed at which the solver finds better solutions over time.

The data farming analysis requires the factors and responses to be consolidated into a single file so that we can explore the input-output relationships. SimOpt facilitates this by automatically creating a comma-separated value (.csv) file suitable for analysis. Each of the $49 \times 2 \times 10 = 980$ rows in this file provides the factor settings, response measures, and any other constants, outputs, or diagnostics recorded for a single run; we call this the *raw* datafile. We also create a summary datafile by aggregating across macroreplications at each design point, obtaining the mean and standard deviation of each numeric-valued response and the probability of achieving each threshold of interest; the *summary-by-problem-and-solver-design-point* datafile has $49 \times 2 = 98$ rows. When analyzing ASTRO-DF's performance on either SAN or SSCONT, we will use only the subset of the rows of the summary-by-problem-and-solver-design-point corresponding to that problem. We can further summarize the data over the "Problem" factor to obtain a *summary-by-solver-design-point* datafile with 49 rows each containing summary statistics for a single combination of solver factors. Some of the graphs and statistical methods that follow use the raw datafile, others are based on one of the summary datafiles.

5.4 Steps in Analysis

Our objective in this section is to provide examples of the types of analyses that might be of interest to a solver specialist or generalist.

5.4.1 Question 1: What was the spread of solver performance metrics over the entire experiment? There are several reasons for asking this question. From the raw datafile, we can determine whether or not all the metrics are available, and whether the ranges of values are plausible. From



(a) $\nu(1)$ (980 obs) (b) A (980 obs) (c) $\tau(0.5)$ (960 obs) (d) $\tau(0.2)$ (805 obs) (e) $\tau(0.1)$ (615 obs) (f) $\tau(0.05)$ (382 obs)

Fig. 3. Histograms and boxplots for several quantitative ASTRO-DF solver performance metrics from the raw datafile. The numbers of non-missing responses are also shown in parentheses. Note that the $\tau(\alpha)$ in 3(c)–3(f) are on the same scale.

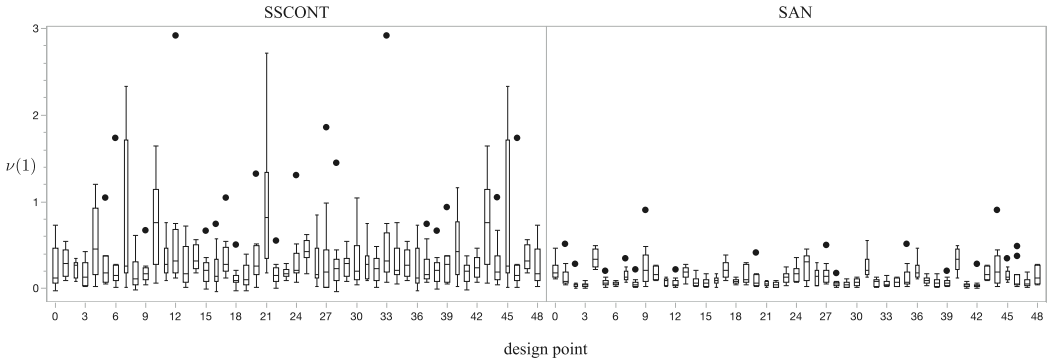


Fig. 4. Boxplots of final relative optimality gap, $\nu(1)$, across design points for two problems (SSCONT and SAN) from the raw datafile.

the summary datafile, we can determine whether the ranges of summary metrics across design points differ enough that we would be interested in fitting metamodels of the response surfaces. For example, Figure 3 shows a series of histograms for the six numeric responses. Additionally, the α-solvabilities (not shown) are $y(0.05) = 0.365$, $y(0.1) = 0.620$, $y(0.2) = 0.837$, and $y(0.5) = 0.965$, respectively; the rows for which the α-solve times are missing correspond to runs where the solver failed to α-solve the problem.

The histograms for the final relative optimality gap and the area under the progress curve were, at first, surprising, because we did not expect any normalized values to fall outside the range $[0, 1]$. After further discussion, we realized this is indeed plausible. We leave a full discussion to Question 8, but remark here that solver generalists and specialists should be aware of the possibility of some solver performance metrics falling outside the normalized limits. The distributions of the α-solve times conform to our expectations that it becomes more difficult to solve problems as α decreases. This is revealed both by the increasing frequencies of runs that use high proportions of the available time, as well as decreasing numbers of runs where the $\tau(\alpha)$ values were reported. Figures 3(c)–3(f) reveal that a greater preponderance of runs require more of the available time to solve the problem as the α threshold for optimality decreases.

5.4.2 Question 2: What run-to-run variation (spread, shape, and central tendency) was observed for the solver performance metrics of interest? Did these vary across design points? Answers to these questions may shape further analysis or lead to additional experiments. The boxplots in Figure 4 show that the final optimality gap data for a specific design point tends to be either

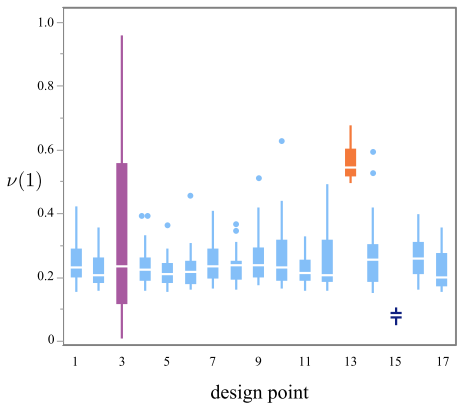
relatively symmetric or positively skewed (towards high values). This is unsurprising since the unknown absolute optimality gap is bounded below by zero. However, it is interesting to see how the distributions of $\nu(1)$ differ by problem: SSCONT is much harder to solve than SAN, which may be due to a small budget chosen for solving it and the problem's high degree of simulation error (variability) when evaluating solutions. The high variability in $\nu(1)$ for SSCONT is not necessarily a cause for concern. A generalist will be more interested in solver performance across a wide range of problems than in predicting the actual solver performance for a specific problem, and there is no requirement that the variabilities across problems be similar. If the generalist was concerned about the number of runs resulting in $\nu(1) > 1$, they could choose to rerun the SSCONT experiments with a larger budget before conducting more detailed analysis. Alternatively, they could add a budget multiplier as a factor in a later experiment; if this ranged from 0.5 to 2.0 (say), then budgets for a specific problem would range from one half to twice their initial budget. This would help ensure that general insight made on solver factors is based on qualitatively similar small, medium, or large budgets, where small, medium, and large could differ by problem. Similarly, a specialist will be more interested in solver performance across a wide range of variants (problem-specific factor combinations) for their problem than in predicting the actual solver performance for a particular variant, but could choose to rerun the SSCONT with a larger budget, or incorporate a budget-related factor into a new experiment, before proceeding.

5.4.3 Question 3: Are there any outliers or solver factor configurations that lead to unusual behavior? Recall that the raw datafile may contain results from hundreds or thousands of runs. Consequently, it is not surprising that many of the histograms or boxplots will reveal large numbers of outliers, in one or both of the typical statistical measures: a distance over roughly two standard deviations from the mean, or a distance over 1.5 times the interquartile range from the ends of the box in a box plot.

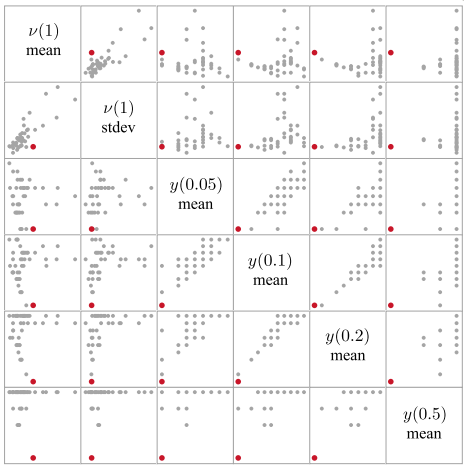
By construction, none of the design points can be considered an outlier in terms of its combination of factor settings, but they can be in terms of the solver performance metrics. While a lot of variability is present across the design points in the ASTRO-DF experiment, none appear to be a clear outlier. In contrast, consider Figure 5(a), where three design points stand out from the rest. In this notional (synthetic) data, design point 3 has a very large spread, design point 13 has a very large median, design point 15 has a very low spread and low median, and the remaining 14 design points have boxplots based on 20 observations generated from essentially identical distributions.

Another graphic that can reveal design points with unusual behavior is a scatterplot matrix of the summary datafile, as shown in Figure 5(b). One design point is interactively selected as an outlier based on the lower-left subplot; this highlights it in all subplots. This selected design point is unusual in that the associated version of ASTRO-DF struggles to achieve even 0.2- or 0.5-solvability. The upper-left subplots indicate that the $\nu(1)$ mean for this design point is fairly high while its standard deviation is fairly low. In other words, this version of ASTRO-DF consistently fails to make good progress on the SSCONT problem. We chose to show plots for a single problem in Figure 5(b), but similar plots could be of interest to a generalist if they were constructed from the summary-by-solver-design-point datafile.

Figures 5(a) and 5(b) clarify another important point: unlike situations involving observational data, unusual design points should not be discarded or ignored. Tracing back to their source may identify bugs in the solver or the solver-problem interface that can be corrected, i.e., the data farming approach can help verify the solver source code. Additionally, unusual design points with undesirable behaviors may help both generalists and specialists set boundaries or limits on the solver factors, while unusual design points with desirable behaviors may help them select appropriate defaults.



(a) Boxplots of the final relative optimality gap $\nu(1)$ across design points for a fictitious solver on a notional problem, from a raw datafile.



(b) Scatterplot matrix for six solver performance metrics from the SSCONT portion of the summary-by-problem-and-solver-design-point datafile. The same design point is highlighted in all subplots.

Fig. 5. Examples of two types of plots that may assist in identifying unusual design points.

5.4.4 Question 4: How are the solver performance metrics correlated? Do they involve tradeoffs or behave serendipitously? This question can be of interest for several reasons. First, at times, it may streamline the detailed analysis process. If, for example, two or more of the solver performance metrics have very strong pairwise correlations, then examining one (rather than all) of them in greater detail for questions 5–10 may be sufficient. Second, the sign of the correlation between two solver performance metrics provides a partial indication about whether improving one will tend to be serendipitous (benefiting the other) or involve tradeoffs (worsening the other).

Figure 6 shows a correlation matrix of the means and standard deviations of the 10 solver performance metrics. (Because the observations of the α-solvabilities are binary responses, their standard deviations are omitted.) One indication of serendipity in the figure is the strong positive correlation between the $\nu(1)$ mean and $\nu(1)$ standard deviation, because the ideal value for both metrics is 0; consistently and rapidly reducing the relative optimality gap is desirable. The strong negative correlation between the $\nu(1)$ mean and the $y(0.05)$ mean is also serendipitous because the ideal value for $y(0.05)$ is 1. The very weak positive correlation between the $\nu(1)$ mean and the $y(0.5)$ mean indicates a slight tradeoff between solver quality and solver speed: solver versions that quickly cut the relative optimality gap in half were slightly more apt to wind up with a poorer final solution. We defer our discussion of the interesting pattern in the 4×4 highlighted box on the right, along with the gray boxes in the lower triangle, until Question 8.

5.4.5 Question 5. Which solver factors are most influential? One way to answer this question is to use partition tree metamodels. We follow the approach in [30] and create a partition tree with 10 splits for each summarized solver performance metric and for each problem in the summary-by-problem-and-solver-design-point datafile. We then multiply the R^2 for each tree by the respective column contributions and categorize the resulting influence as **very very strong (VVS)**, **very strong (VS)**, **strong (S)**, **moderate (M)**, **weak (W)**, and **very weak (VW)**. These characterizations are determined by inverse powers of two, denoting contributions to R^2 that are > 0.50, > 0.25, > 0.125, > 0.0625, > 0.03125, and > 0.015625, respectively. This automated heuristic method limits the number of factors that will be deemed influential without requiring

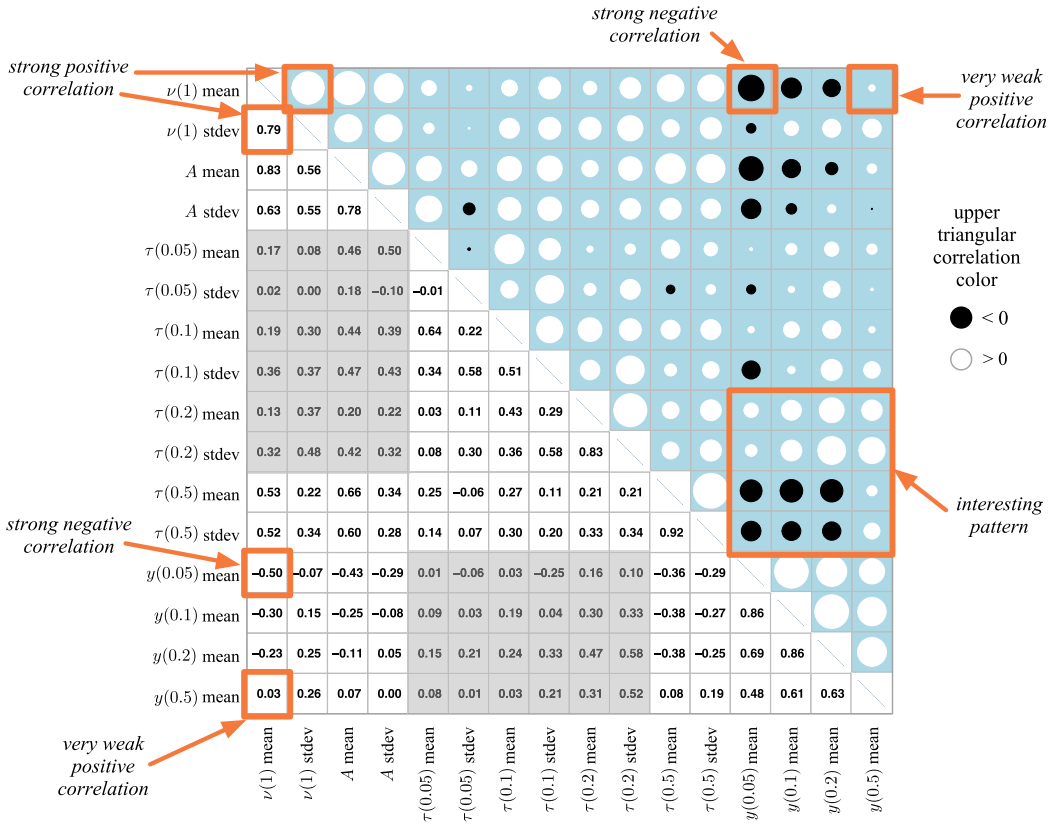


Fig. 6. Numeric and graphical correlation summaries for a generalist, from the summary-by-solver-design-point datafile. Numerical correlations are provided in the lower triangular region. In the upper triangular region, white circles represent positive correlations, black circles represent negative correlations, and a circle's area represents the strength of the correlation.

detailed human-in-the-loop assessments by the analyst. Our results, broken out by problem, appear in Table 4. Recall that the first six responses are related to the solution quality, while the last ten are related to the speed at which the solution is found. The R^2 values in Table 4 are sufficiently high that at least one solver factor is deemed influential, either exhibiting VS or VVS influence. If R^2 were low for some other solver performance metric, we could have seen few (if any) factors showing up with more than medium influence.

An inventory specialist would be interested in the SSCONT results from Table 4. For this experiment, it is clear that γ_1 was the most influential factor, with VVS effects for 12 of the 16 solver performance metrics and VS or S for the remaining four. The factor γ_2 is also very influential, with medium to very strong effects on all but four of the responses. The factor η_1 was much less influential, but had an impact on a few solvability metrics and several metrics related to the solver's speed. The factor η_2 was the least influential, with none of the factors influencing any of the solution quality metrics, while contributing to only two of the ten speed-related metrics. The inventory specialist's next step might be to examine more closely the leaves of the partition trees to determine desirable settings for ASTRO-DF's factors—particularly γ_1 and γ_2. One nice characteristic of partition tree metamodels is that they make few assumptions about the nature of the influence. For example, increasing an influential factor could result in increasing or diminishing returns on

Table 4. Summary of Factor Influences on ASTRO-DF Performance Metrics

	SSCONT					SAN				
Response	η_1	η_2	γ_1	γ_2	R^2	η_1	η_2	γ_1	γ_2	R^2
$v(1)$ mean			VVS	VS	0.859		M	M	VVS	0.780
$v(1)$ stdev			VVS	VS	0.771	M	M	W	VVS	0.896
$y(0.05)$			VVS	S	0.910			VVS	VS	0.876
$y(0.1)$	VW		VVS	VS	0.901		W	VS	VVS	0.909
$y(0.2)$	M		VS	M	0.875	VW	W	S	VVS	0.606
$y(0.5)$			VVS		0.922			VVS	VS	0.740
A mean	S		VVS	S	0.908	W		VS	VVS	0.930
A stdev			VVS	W	0.724	M	M		VVS	0.948
$\tau(0.05)$ mean			VVS	S	0.876			VVS	VS	0.910
$\tau(0.05)$ stdev		S	S	VS	0.713		W	VVS	S	0.853
$\tau(0.1)$ mean	VW		VVS	VS	0.909		W	VS	VVS	0.901
$\tau(0.1)$ stdev			VVS	S	0.880	S	W	S	VS	0.789
$\tau(0.2)$ mean	M		VS	M	0.606	VW	W	S	VVS	0.875
$\tau(0.2)$ stdev	W	W	S	VS	0.713		S	VS	VS	0.904
$\tau(0.5)$ mean			VVS		0.740			VVS	VS	0.922
$\tau(0.5)$ stdev			VVS	W	0.600		VW	VS	VS	0.800

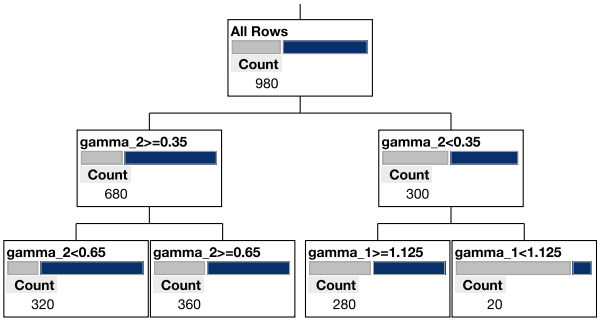
Classifications are determined by inverse powers of 2, and are categorized as very very strong (VVS), very strong (VS), strong (S), medium (M), weak (W) and very weak (VW).

a solver performance metric, or there could be a "sweet spot" that is advantageous. A specialist in charge of project management for the SAN problem could likewise draw insights from the SAN results in Table 4.

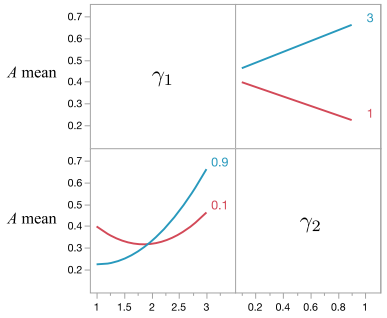
Note that a generalist might be less interested in the specific problem and treat the choice of problem as a noise factor (see Question 9). This corresponds to growing a tree from the summary-by-solver-design-point datafile rather than growing two separate trees from the SAN and SSCONT portions of the summary-by-problem-and-solver-design-point datafile. When growing a single tree, the "Problem" (a qualitative factor) could be included as a potential explanatory variable; this might be worthwhile to a generalist if the solver's performance differs greatly across different types of problems.

Other types of metamodels are possible. For example, stepwise regression can be used to fit metamodels containing low-order polynomial terms. Remember, however, that statistical significance is not the same as practical importance. With the very large datasets that can result from a data farming experiment, the analyst should place a higher emphasis on parsimony. This may mean removing many statistically significant terms from automatically generated models because their relative impact is negligible.

5.4.6 Question 6: Are there any important interactions? Consider the plots in Figure 7. On the left, Figure 7(a) shows the first three layers of the 10-split partition tree for 0.1-solvability. These do not correspond to the first three splits. An interaction between γ_1 and γ_2 is revealed by the difference in splitting on the left and right branches. The light and dark bars in each leaf indicate the proportion of non-solved and solved runs, respectively, with dark bars being preferred. The top level shows that the majority of the runs were able to achieve 0.1-solvability. The middle level shows that larger γ_2 values (slower rate of trust-region contraction) are associated with a greater likelihood of finding good solutions. The third level shows that γ_2 should not be too large, and that if γ_2 is small then it is very important for γ_1 to be large, i.e., if the penalty of not succeeding is a



(a) Partial partition tree for a generalist, showing the top three layers of the 10-split partition tree for the 0.1-solvability, $y(0.10)$, for both problems using the raw datafile. The light and dark bars in each leaf indicate the proportions of non-solved and solved runs, respectively.



(b) Interaction profiles for a specialist, based on stepwise regression metamodel for A mean for the SSCONT problem. There is a quadratic effect for γ_1, a linear effect for γ_2, and a strong interaction.

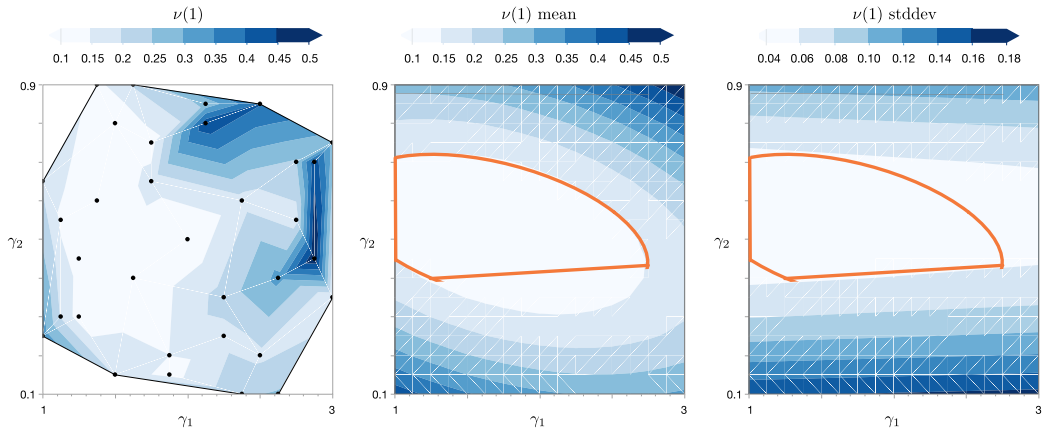
Fig. 7. Two types of graphs for revealing interaction effects from two types of metamodels for different solver performance metrics.

significant contraction, then a success should let the trust region recover its size with a significant expansion.

On the right, Figure 7(b) is a graphical depiction of interaction effects from a regression metamodel for the mean area under the curve (the mean of A) after using stepwise regression, combined with expert judgment, to obtain a parsimonious metamodel. The upper-right box shows that if γ_1 is set to its low value (1.0), then increasing γ_2 leads to a linear decrease in the A mean; if γ_1 is set to its high value (3.0), then increasing γ_2 leads to a linear increase in the A mean. In other words, the best setting for γ_2 depends on the best setting for γ_1, and vice-versa—i.e., there is an interaction. Note, however, that because low values of the A mean are desirable and the two lines do not cross, the better choice for γ_1 is 1. The lower-left box is another snapshot of the multidimensional metamodel. Here we see that after fixing the value of γ_2, γ_1 has a nonlinear (quadratic) effect on the solver performance metric. If $\gamma_2 = 0.9$, then the metric gets consistently worse as γ_1 increases. In contrast, if $\gamma_2 = 0.1$, then the best setting for γ_1 is around 2.0 where the curve achieves its minimum value.

5.4.7 Question 7: What are the interesting regions and threshold values in the solver factor space? As we have seen in Figure 7(a), the leaves of partition trees may themselves indicate interesting regions. As we move from left to right across the leaves, the proportion of runs on which the problem is solved to 0.1 optimality decreases. In particular, below a certain level for γ_2 (the rate of trust-region contraction), solving the problems accurately largely depends on choosing larger γ_1 (the rate of trust-region expansion). While the latter observation aligns with a general guideline of choosing $\gamma_2 = \gamma_1^{-1}$, our results show that the choice of γ_2 is more important (rather than setting γ_1 and taking the reciprocal), and the inverse relationship may not be accurate or of much help, depending on the choice of γ_2. Although this insight matters to a generalist, the findings in Table 4 suggest that the importance of γ_2 over γ_1 is more problem-dependent; e.g., for the SSCONT problem, all solver performance metrics are more sensitive to the choice of γ_1 than γ_2.

Another possibility is a contour plot; an example that might be of interest to a generalist (i.e., one not broken out by application) is shown in Figure 8(a). This plot is based on the raw datafile, but is identical to that based on the summary-by-solver-design-point datafile because it makes use of the mean values at each point. We remark that for experiments involving large numbers of factors, Questions 5 and 6 may indicate which pairs of factors to select for contour plots for



(a) Contour plot of $\nu(1)$ vs. two influential factors γ_1 and γ_2 using the raw datafile for both SAN and SSCONT.

(b) Contour plot of the $\nu(1)$ mean vs. γ_1 and γ_2 from a regression metamodel fit to summarized data for SAN.

(c) Contour plot of $\nu(1)$ standard deviation vs. γ_1 and γ_2 from a regression metamodel fit to summarized data for SAN.

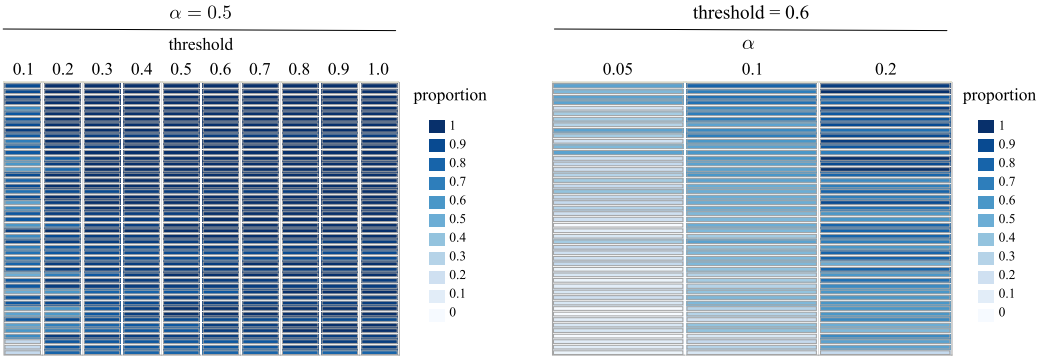
Fig. 8. Contour plots. A generalist might be interested in plot (a), while plots in (b) and (c) might be of interest to a specialist. The regions highlighted in (b) and (c) indicate where both solver performance metrics are favorable for SAN.

specific solver performance metrics, since considering all possible pairs may be prohibitive. The contour plots in Figures 8(b) and 8(c) depict metamodels of the mean and standard deviation of the final relative optimality gap for the SAN problem and appear smoother than the plot in Figure 8(a) because they are based on regression metamodels. The highlighted region in Figures 8(b) and 8(c) is one where good results are obtained for both solver performance metrics. A specialist might be interested in a follow-up experiment that focuses more closely around this region.

5.4.8 Question 8: Are any of the results counterintuitive? A counterintuitive finding can be either a bug or a feature. Large-scale data farming experiments can expose behaviors that would otherwise go unnoticed. On further inspection, one of two things might happen. First, if the finding can be traced back to its cause and shown to be a bug (either a miscoding or an inappropriate assumption), this bug can be corrected. This is part of the verification process. Alternatively, the finding might withstand further scrutiny, and so change our intuition. This is part of the validation process.

A few counterintuitive results appeared earlier. In Question 1, we remarked that some runs yielded $\nu(1)$ values that were less than 0 or much greater than 1. The explanation for this is twofold. Because the optimal solution is unknown, we rely on an estimate of the response mean based on postreplications. This means that if several recommended solutions have similar near-optimal objective function values, and the postreplications on the observed best solution improve its estimated solution value, that can lead to small negative numbers reported for the final relative optimality gap. On the other hand, a final relative optimality gap greater than 1 can occur if the solver recommends a solution that it believes to be better than x_0—based on its limited sampling at said solution—when in fact the solution is worse. (More commentary on these situations is given in [8].) The same reasoning explains why values of A outside of the interval $[0, 1]$ are possible.

Another example of seemingly counterintuitive results is the 4×4 box in Figure 6 highlighted for its interesting pattern. Before looking at the correlation plot, we anticipated that for any given α, the $\tau(\alpha)$ mean and the $y(\alpha)$ mean would be negatively correlated, because finding a satisfactory solution quickly corresponds to low $\tau(\alpha)$ and high $y(\alpha)$. We also anticipated that the strength



(a) Heatmap of the proportion of runs for which $\tau(0.5) \leq$ threshold for various budget thresholds.

(b) Heatmap of the proportion of runs for which $\tau(\alpha) \leq 0.6$.

Fig. 9. Heatmaps of the $\tau(\alpha)$ for a generalist created from the summary-by-solver-design-point datafile. Each row corresponds to a single design point, and the cell shadings reflect how often $\tau(\alpha)$ satisfied certain conditions. In (a), where little censoring occurs at the end of the run (threshold = 1), almost all the solver design points are highly likely to reach 0.5-solvability early in the run. In (b), where a great deal of censoring occurs, we observe how moving from left to right (increasing α) increases how likely it is to achieve α-solvability within a fixed budget threshold.

of the correlation would be highest when $\tau(\cdot)$ and $y(\cdot)$ used the same α, rather than different αs. The highlighted box shows patterns for $\tau(0.2)$ that differ in both the sign and the magnitude of correlation from those of $\tau(0.5)$. However, upon further investigation we realized this was an artifact of the different numbers of design points used to compute the correlations due to censoring. We constructed the correlation plot in Figure 6 using all available data, but recall that when the solver cannot α-solve the problem within the allowable budget in a run, then that run's $\tau(\alpha)$ is a missing value and the run is not part of correlation computations between the $\tau(\alpha)$ mean or variance and the $y(\alpha_1)$ for all α_1.

Constructing the correlation plot was useful to us, because it alerted us to the censored data issue. In such cases, the analyst should be aware that some correlations will not be easily interpretable. In our experiment, the grayed-out rectangles in the lower triangle correspond to situations where between 16.3% and 63.5% of the runs had missing τ data. In contrast, if we had used sufficiently large budgets that all the runs were solved to $\alpha = 0.05$, then $y(\alpha) \equiv 1$ for all $\alpha \geq 0.05$ and all correlations between a $\tau(\alpha_1)$ mean or standard deviation with a $y(\alpha_2)$ would equal zero.

How, then, should we assess the joint behavior of the $y(\alpha)$ and $\tau(\alpha)$? Both are important, and both provide different information than can be gleaned from the $v(1)$ and A means and standard devations alone. A better way of showing the relationships involving censored data is via the use of heatmaps, such as those shown in Figure 9. Each row in the heatmaps corresponds to a single design point in the summary-by-solver-design-point file. The columns specify conditions on at least one solver factor or metric, and the color of each cell corresponds to the proportion of runs for which the conditions are satisfied for the solver performance metric of interest (light=low, dark=high).

Heatmaps such as that in Figure 9(a) are useful when few, if any, $\tau(\alpha)$ values are censored for a particular α. The columns correspond to different budget thresholds for this α. In these situations—especially if α is near zero—it may be of interest to calculate how likely the solver is to achieve a specified performance without exceeding smaller thresholds (proportions) of the normalized budget. We calculated these values for thresholds of 0.1 to 1.0 in steps of 0.1. This may help the

analyst determine whether or not a smaller budget would suffice; in our experiment, all the solver design points are 0.5-solvable after expending only a small portion of their budget. Conversely, if many $\tau(\alpha)$ values are censored for a particular α, then a heatmap such as that in Figure 9(b) is more informative. The columns correspond to different α for a particular threshold. It may help the analyst determine whether or not accepting a larger α would mitigate the censoring issues without requiring larger budgets, which could be of particular interest if (near) real-time solutions are needed. Note that in either heatmap, we do not need all of the rows to exhibit good performance, just some of the rows.

5.4.9 Question 9: Which solver factor configurations are most robust, leading to consistently good performance for a solver metric of interest? Although this is one of the later questions, answering it is very important in the context of simulation optimization. Both generalists and specialists seek settings for the solver factors that produce consistently good solver performance metrics across a range of problems, regardless of the problem-specific factors or data characteristics. This can be accomplished by incorporating two types of factors into the experiments: *decision factors* that are controllable, and *noise factors* that are controllable within the simulation experiment, but not controllable or controllable only at great cost in a more general setting. In our context, the noise factors for a specialist could consist of problem-specific factors in Table 3; the noise factors for a generalist could be as limited as a list of different problems, factors that describe problem characteristics such as dimensionality and convexity, or a mixture of general and problem-specific factors. The experiment design can be constructed by including all factors (solver factors, the categorical problem factors, and problem-specific factors) in a single design for greater efficiency, or by crossing separate designs for decision and noise factors, to facilitate decision-factor comparisons without requiring metamodels.

After the solver experiments have been conducted, a loss function can quantify how well the solver performed with respect to a particular solver performance metric. Squared-error loss relative to an ideal outcome (a.k.a. the *target value*) is often useful; outcomes close to the target contribute little to the overall loss, while those far from the target contribute a great deal. For example, the ideal (though unachievable) value for A is $A^* = 0$. A squared error loss of the form $loss_A = (A-A^*)^2$, described in Section 2.2, means that the expected loss of a given solver version (aggregated over the noise factor space) decomposes into the form $\mathbb{E}[loss_A] = (\mathbb{E}[A] - A^*)^2 + \mathrm{Var}(A)$. This is one reason why we considered both the means and standard deviations when summarizing our solver performance metrics. Metamodels fit to the mean and standard deviation of A, for instance, can be used to determine if there are solver factor settings that yield results consistently near zero. Similarly, the ideal value for $\nu(1)$ is $\nu(1)^* = 0$, which might be achievable for sufficiently large computing budgets. Table 5 summarizes the characteristics of low-order polynomial metamodels fit to the mean, standard deviation, and loss associated with $\nu(1)$ for specialist and generalist perspectives. It also provides the configurations that are predicted to provide the best performance for each criterion, and the avoidable loss associated with using the non-low loss configurations. For the SAN problem, the most impactful terms involved γ_1 and γ_2: the configurations projected to yield the best terminal solution on average suggested a higher value for γ_1 (fast expansion of step size), whereas the setting $\gamma_1 = 1$ was predicted to be better for minimizing the variability of the terminal solution. Metamodels for the SSCONT specialist differed and had lower explanatory power. A low expansion rate γ_1 was best for all three metamodels. The only other impactful factor was γ_2, where an interaction effect leads to tradeoffs between achieving a good mean (high γ_2) or a low variance (low γ_2). The generalist metamodels also had low explanatory power, i.e., robustness across these two problems is difficult to achieve. Consequently, this illustrates that a SAN specialist would benefit from choosing solver settings from a data farming experiment

Table 5. Performance of Polynomial Metamodels for Identifying Good Parameter Settings

User	Criterion to Minimize	Metamodel R^2	# terms M	Q	I-2	Best for Criterion Configuration $(\gamma_1, \gamma_2, \eta_1, \eta_2)$	Predicted Value	Predicted Loss	Avoidable Loss
Specialist (SAN)	Mean($\nu(1)$)	0.79	2	2	1	(1.74, 0.51, ·, ·)	0.000	[-0.024, 0.001]	[0.005, 0.036]
	Stdev($\nu(1)$)	0.58	4	2	1	(1.0, 0.50, 0.1, 0.5)	-0.010	-0.020	0.005
	Loss($\nu(1)$)	0.93	4	3	2	(1.63, 0.49, 0.1, 0.5)	-0.025	-0.025	-
Specialist (SSCONT)	Mean($\nu(1)$)	0.35	2	0	1	(1.0, 0.9, ·, ·)	0.103	0.158	0.257
	Stdev($\nu(1)$)	0.34	2	1	0	(1.0, 0.1, ·, ·)	-0.017	-0.099	-
	Loss($\nu(1)$)	0.38	2	0	0	(1.0, 0.1,·, ·)	-0.099	-0.099	-
Generalist	Mean($\nu(1)$)	0.40	2	0	1	(1.0, 0.9, ·, ·)	0.065	0.005	-
	Stdev($\nu(1)$)	0.26	1	0	0	(1.0, ·, ·, ·)	0.052	[0.005, 0.415]	[0, 0.410]
	Loss($\nu(1)$)	0.26	2	0	1	(1.0, 0.9, ·, ·)	0.005	0.005	-

Potential terms are main (M) and quadratic (Q) for each of the four solver factors, as well as the six two-way interaction terms (I-2). The 'Predicted Loss' evaluates the user's 'Low Loss' metamodel at the given configuration; the 'Avoidable Loss' predicts the loss incurred if 'Low Loss' configuration differs.

rather than relying on default values specified by a generalist. The improvement can be dramatic: for the SAN problem, the confirmation runs showed that the recommended configuration yielded a loss of only 0.0077, which was only 66.5% of the loss at the center point (2.0, 0.5, 0.3, 0.7) and less than 0.05% of the loss at a particularly bad configuration in the region (3.0, 0.9, 0.1, 0.9).

Some additional remarks related to the solver parameter tuning: If a metamodel suggests configurations that are not among the original design points, as is often the case, confirmation runs should be made. If the results of these confirmation runs are consistent with the metamodels, all is well. If not, the analyst has several choices. First, they can use this new configuration if they deem sufficient improvement has occurred. Second, they can refit metamodels that incorporate the new data, or fit metamodels of a different type. For example, some of the predictions in Table 5 are negative, indicating that polynomial metamodels may be struggling to adequately fit values near zero. This issue might be mitigated if metamodels were fit to logarithmic transformations of the response criteria, or if different types of metamodels (e.g., partition trees, logistic regression, Gaussian process metamodeling) were used. They can run additional replications of the existing design if they wish to distinguish finer gradations in performance. For example, with $m = 10$ replications, the $y(\cdot)$ metrics can take on only $m + 1$ potential values (0, 0.1, 0.2, …, 1.0); a larger m may be needed to differentiate configurations with underlying means of, say, 0.98 and 0.96. Finally, they might use the information they have gained to run a second experiment before settling on a suitable configuration.

Many of the graphical methods discussed in previous sections can also be used. For example, the comparative box plots in Figure 5(a) clearly show that design point 15 is the most robust; contour plots such as those in Figures 8(b) and 8(c) can be used if a solver performance metric's mean and standard deviation are the two solver performance metrics of interest. For more details on robust design in the simulation context, see [40] and [42]. Besides using the aforementioned loss function to capture the robustness of a solver version, one can use VaR(A) or CVaR(A).

5.4.10 Question 10: Are there any Configurations that Perform well w.r.t. Multiple solver performance metrics? We presented multiple solver performance metrics and discussed them throughout this paper. The mean and variance of the final relative optimality gap $\nu(1)$, together with the mean α-solvabilities for various αs, measure the quality of the solver's solutions; the means and variances of the area under the progress curve, A, and α-solve times, $\tau(\alpha)$, measure the speed at which better and better solutions are identified.

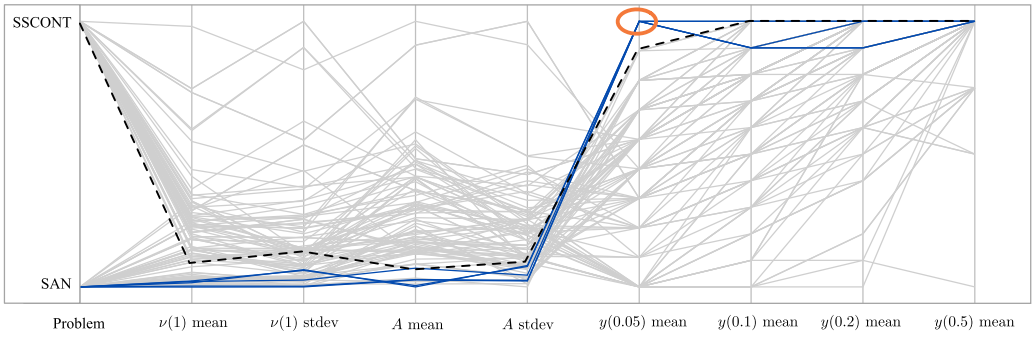


Fig. 10. Parallel plot for a generalist. Each of the 98 traces represents summarized output for eight solver performance metrics from a single row of the summary-by-problem-and-solver-design-point datafile. The minimum and maximum values for each solver performance metric are the lowest and highest in this datafile, respectively. Low values are desirable for the first four solver performance metrics involving $\nu(1)$ and A, while high values are desirable for the last four. Traces corresponding the highest observed $y(0.05)$ mean are indicated with dark solid lines and come from the SAN problem. The dashed line indicates a solver design point that performs well for SSCONT.

Clearly there are multiple performance measures of interest to either generalists or specialists who decide to conduct data farming to tune solver factors. We have shown in multiple ways that our primary solver performance metrics capture different aspects of the solution quality and the speed at which it is achieved: no solver configuration dominates on all measures (Figure 1(b)); correlations may mean that improving one solver performance metric is detrimental to another (Figure 6); and the sets of most influential factors or interactions differ across the solver performance metrics (Table 4). Robust analysis provides a quantitative approach for making tradeoffs among means and standard deviations of quantitative solver performance metrics. Assigning weights to loss functions, if feasible, can similarly help identify solver factor settings that are robust (in terms of minimizing a weighted multi-objective loss) across different performance measures [44]. Heatmaps could include columns for more solver performance metrics, potentially with their own sets of thresholds.

A few other approaches are worth mentioning. One is to add user-specified constraints for multiple solver performance metrics, and create a new column in the appropriate raw or summary datafile indicating whether a particular row simultaneously satisfies all of the constraints. To the extent that metamodels can do a reasonable job of capturing the relationship between solver factors and such a binary response, this will help set solver factor defaults or provide guidance about modifying the defaults in different settings. We can also explore this graphically, as with the parallel plot in Figure 10 based on the summary-by-problem-and-solver-design-point datafile. Eight solver performance metrics are shown, with the y-axis scaled separately for each metric between its minimum and maximum in the raw datafile. We used coded numeric values so we could include the problem in this plot. Each trace in the plot corresponds to one of the 98 design points. Highlighting the lines that fall within the highest observed $y(0.05)$ in Figure 10 reveals that these design points yield good performance on the other seven measures as well. Note also that they all correspond to the SAN problem. The dashed line indicates a design point that works well on all measures for SSCONT. A second is to identify a Pareto front, providing a set of solver parameter settings from which the generalist or specialist can choose, depending on which solver performance metrics they deem most important. If the solver factors are continuous, then spreading the choices in the factor space will keep the

computational requirements manageable and provide the decision maker with a set of distinct alternatives [6, 46].

5.5 Discussion

We have seen that a wealth of information can be gleaned from the data produced by a design of experiments over solver factors. The "Top Ten" questions can help to guide the analysis about what to look for when evaluating solver performance and how to visualize the results. From even just a rudimentary experiment that farmed four factors controlling trust-region expansions and contractions, we gained new insights into the behavior of ASTRO-DF on two SO problems. This included observing that on these problems, rapid progress and run-to-run consistency are not mutually exclusive qualities and that the settings of the expansion rate γ_1 and contraction rate γ_2 have a strong influence on the solver's progress, with there being an interaction between them. We found that, depending on the problem, one of these factors may be more influential than the other, but thresholds at which to expand or contract the trust region (i.e., η_1 and η_2) are less critical in both problems.

As stated earlier, our goal for this article was to propose and illustrate a data farming approach for investigating solver performance, rather than to present a definitive study of ASTRO-DF. Consequently, we conducted a small experiment involving only four solver factors and only two problems to illustrate the methods. This can serve as a template for future studies of ASTRO-DF or other solvers by generalists, as well as future applications by specialists. Two points are particularly important. First, the ability to use large-scale space-filling experiment designs breaks the so-called "curse of dimensionality" and fundamentally changes the types of studies that can be conducted. Efficient designs, such as those referenced in Section 5.3, enable generalists and specialists to include dozens or hundreds of factors in their studies, rather than a handful or fewer. Second, adopting this new mindset fundamentally changes the types of insights that can be gained. Solver benchmarks can be based on massive sensitivity analyses of solver parameters, rather than default configurations alone. Solver generalists can seek robust parameter settings, and provide better guidance to users about when and how to modify these settings, by identifying the I/O relationship between factors and solver performance metrics. Solver specialists can employ data farming not just to tune parameters for their particular application, but to develop insights and guidance about real-world circumstances that might lead them to change these parameters without needing to perform additional experiments. We assert that insights from these types of studies can be much broader than those obtainable from tuning a limited number of hyperparameters.

The analysis carried out in this section was for a non-adaptive experiment, wherein the factors of interest and their ranges were specified up front, a design was generated, and a common number of runs were performed at each design point. However, the data-farming approach can and arguably should be applied as an iterative process. There is an emerging interest is producing tuning algorithms that adjust the algorithm hyperparameters during the course of the optimization, as knowledge about local or global behavior of the problem is collected. The goal of these so-called online tuners or parameter control methods [25] is well within the potential capabilities that our proposed framework offers. It is often good practice to start with a preliminary design that includes many factors and, from the results, identify those that most influence solver performance. In early stages of solver development, this approach assists in verifying the implementation by subjecting the solver to a massive sensitivity analysis, where the analysis questions are much more likely to uncover bugs in the code than a trial-and-error approach or more limited experimentation. In later stages, solver factor ranges can be refined or a crossed design (where solver factors deemed unimportant could be relegated to the noise factor design) can help to focus the investigation on the important ones. The initial design can also be

refined by adding stacks to fill under-covered regions of the solver factor space and additional runs can be performed at design points with highly variable solver performance metrics.

The proposed framework can also be used in ways other than what we have demonstrated. While the analyses can help to build a deeper understanding of how a solver's factors affect its empirical performance, they can also shed light on questions about the design of the solver itself. Such questions could be of practical importance, such as the need (or lack thereof) for particular operations within the solver, or theoretical importance, such as how a solver's factors might need to change adaptively throughout a run to ensure convergence. Specifically, for trust-region methods like ASTRO-DF, a generalist might wish to assess the benefits of various designs for sampling solutions within the trust region or to determine whether the thresholds η_1 and η_2 should be updated over time.

A specialist may instead be interested in using the framework for tuning the solver, which refers to finding solver parameter settings that deliver good performance on a given problem or class of problems. If the user needed to solve only a single problem, then a small design over solver factors could even be conducted *within* the solver, in a preliminary phase, by running a few pilot runs with a fraction of the budget before committing to a configuration of solver factors. If the user instead needed to solve a sequence of similar problems, then a larger offline design of experiments could help identify solver factor configurations that are robust to variations in the problem factors. The task of finding the best configuration of solver factors could be carried out by fitting a metamodel to one or more solver performance metrics, e.g., the mean and standard deviation of the final relative optimality gap $\nu(1)$, and optimizing.

6 Conclusion

We introduce a data-farming framework for systematically studying the performance of SO solvers as a function of their parameters, which we term solver factors. In particular, we explore how a design over solver factors can be used to investigate their influences on multiple solver performance metrics measuring the rate of improvement in the recommended solutions and the quality of the solution ultimately recommended. The framework offers a flexible and powerful means of exploring important questions of interest to a range of stakeholders, from solver developers to end users, from solver generalists to specialists. Moreover, the framework leverages efficient designs to allow users to holistically investigate solver performance under different parameter settings in a way that goes beyond hyperparameter tuning or benchmarking solvers with default parameters. We advocate for solver developers to embrace DOE as a tool for testing and improving their solvers prior to deployment, and for specialists to embrace it to improve the timeliness of efficacy associated with repeatedly optimizing their simulation applications.

In this article, we focused on solver factors that remain fixed throughout the solver's process of finding better and better solutions. However, the framework permits the study of solver factors that change *adaptively* and can help devise schedules or conditions for updating them that yield strong solver performance. Categorical factors that fundamentally alter the operations of a solver, such as the type of local model (linear versus quadratic) constructed within a trust-region solver, can also be varied. Although we varied only solver factors in this article, the framework can be extended in several noteworthy directions by incorporating other experiment parameters as factors in the design. For example, the initial solution x_0 and the budget T could be varied to assess how the solver's long-term behavior is affected; e.g., does it converge to different local optima when starting in different regions? In addition, factors of the simulation model can be varied, as is typically done in data farming experiments, to illuminate how robust a solver is at finding good solutions when the simulation model is subject to misspecification. This poses an additional level of uncertainty, called input uncertainty, that must be factored into the solver performance metrics.

More broadly, a solver can be tested over a large set of problems exhibiting different properties, including the dimension or geometry of the feasible region and the structure of the objective function, as proposed by [48]. The ultimate goal of such a study would be to develop guidelines for users when setting the factors of a solver that is to be run on a specific problem.

Further research in this space will be accompanied by enabling the creation and execution of designs in the SimOpt testbed [10]. This will involve developing a graphical user interface for users to first set up a design—by selecting the problems and solvers and the factors they wish to include—and then run the designed experiment. (Within SimOpt, such experiments can be accelerated by using multithreading to perform solver runs in parallel.) The final product will be an output file containing the specifications of each design point and the solver performance metrics discussed in this article. This file can then be loaded into a user's preferred statistical software package for subsequent analysis. We also aim to add more solvers and problems to the library to enable more wide-ranging experiments. Ultimately, we hope to encourage a fuller study of well-established solvers to see what more can be learned about them and what guidance can be provided to end users.

References

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems* 24 (2011).

[2] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 2 (2012).

[3] Mauro Birattari, Thomas Stützle, Luis Paquete, Klaus Varrentrapp, et al. 2002. A racing algorithm for configuring metaheuristics. In *Gecco*, Vol. 2. Citeseer.

[4] John R. Birge and Francois Louveaux. 2011. *Introduction to Stochastic Programming*. Springer Science & Business Media, New York.

[5] Mark Broadie, Deniz Cicek, and Assaf Zeevi. 2011. General bounds and finite-time improvement for the Kiefer-Wolfowitz stochastic approximation algorithm. *Operations Research* 59, 5 (2011), 1211–1224.

[6] Licun Edwin Cai. 2018. *Exploring Characteristics of an Effective Mix of Precision and Volume Indirect Fire in Urban Operations using Agent-based Simulation*. Master's thesis. Naval Postgraduate School, Monterey, CA.

[7] Thomas M. Cioppa and Thomas W. Lucas. 2007. Efficient nearly orthogonal and space-filling latin hypercubes. *Technometrics* 49, 1 (2007), 45–55.

[8] David J. Eckman, Shane G. Henderson, and Sara Shashaani. 2023. Diagnostic tools for evaluating and comparing simulation-optimization algorithms. *INFORMS Journal on Computing* 35, 2 (2023), 350–367.

[9] David J. Eckman, Shane G. Henderson, and Sara Shashaani. 2023. SimOpt: A testbed for simulation-optimization experiments. *INFORMS Journal on Computing* 35, 2 (2023), 495–508.

[10] David J. Eckman, Shane G. Henderson, Sara Shashaani, and Raghu Pasupathy. 2020. Simulation Optimization Library. http://github.com/simopt-admin/simopt. (2020). [Online; Accessed May 1, 2020].

[11] J. A. Egea, D. Henriques, T. Cokelaer, A. Villaverde, A. MacNamara, D.-P. Danciu, J. R. Banga, and J. Saez-Rodriguez. 2014. MEIGO: An open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. *BMC Informatics* 15 (Article 136 2014). https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-15-136

[12] C. Erickson, B. E. Ankenman, and S. M. Sanchez. 2018. Comparison of Gaussian process modeling software. *European Journal of Operational Research* 266, 1 (2018), 179–192.

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.

[14] Peter I. Frazier. 2018. Bayesian optimization. In *Recent Advances in Optimization and Modeling of Contemporary Problems*. Informs, 255–278.

[15] Aldy Gunawan, Hoong Chuin Lau, and Lindawati. 2011. Fine-tuning algorithm parameters using the design of experiments approach. In *5th International Conference on Learning and Intelligent Optimization (LION 5), (*Rome, Italy, January 17-21, 2011*)*. Selected Papers 5. Springer, 278–292.

[16] Yunsoo Ha and Sara Shashaani. 2023. Iteration complexity and finite-time efficiency of adaptive sampling trust-region methods for stochastic derivative-free optimization. *arXiv preprint arXiv:2305.10650* (2023).

[17] Yunsoo Ha, Sara Shashaani, and Quoc Tran-Dinh. 2021. Improved complexity of trust-region optimization for zeroth-order stochastic oracles with adaptive sampling. In *2021 Winter Simulation Conference*, S. Kim, B. Feng, K. Smith,

S. Masoud, Z. Zheng, C. Szabo, and M. Loper (Eds.). Institute of Electrical and Electronics Engineers, Inc., Piscataway, N. J., 1–12. https://doi.org/10.1109/WSC52266.2021.9715529

[18] L Jeff Hong, Zhaolin Hu, and Guangwu Liu. 2014. Monte carlo methods for value-at-risk and conditional value-at-risk: A review. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 24, 4 (2014), 1–37.

[19] L Jeff Hong and Xiaowei Zhang. 2021. Surrogate-based simulation optimization. In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*. INFORMS, 287–311.

[20] Changwu Huang, Yuanxiang Li, and Xin Yao. 2019. A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation* 24, 2 (2019), 201–216.

[21] Deng Huang, Theodore T. Allen, William I. Notz, and Ning Zeng. 2006. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization* 34 (2006), 441–466.

[22] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference (LION 5)*, (Rome, Italy, January 17-21, 2011). *Selected Papers 5*. Springer, 507–523.

[23] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Kevin P. Murphy. 2009. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *11th Annual Conference on Genetic and Evolutionary Computation*. 271–278.

[24] Frank Hutter, Holger H. Hoos, and Thomas Stützle. 2007. Automatic algorithm configuration based on local search. In *AAAI*, Vol. 7. 1152–1157.

[25] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E. Eiben. 2014. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation* 19, 2 (2014), 167–187.

[26] Jeffrey Larson, Matt Menickelly, and Stefan M. Wild. 2019. Derivative-free optimization methods. *Acta Numerica* 28 (2019), 287–404.

[27] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

[28] Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. 2021. Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 343–351.

[29] Wai-Kei Mak, David P. Morton, and R. Kevin Wood. 1999. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* 24, 1-2 (1999), 47–56.

[30] David O. Marlow, Susan M. Sanchez, and Paul J. Sanchez. 2019. Testing policies and key influences on long-term aircraft fleet management using designed simulation experiments. *Military Operations Research* 24, 3 (2019), 5–25.

[31] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2016. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning. Proceedings of Machine Learning Research*, 58–65.

[32] Jorge J. Moré and Stefan M. Wild. 2009. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* 20, 1 (2009), 172–191.

[33] Volker Nannen and Agoston Endre Eiben. 2006. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *8th Annual Conference on Genetic and Evolutionary Computation*. 183–190.

[34] NATO. 2014. *Data Farming in Support of NATO*. Technical Report TR-MSG-088. NATO Science & Technology Organization. https://doi.org/10.14339/STO-TR-MSG-088

[35] David Orive, G. Sorrosal, Cruz Enrique Borges, Cristina Martin, and A. Alonso-Vicario. 2014. Evolutionary algorithms for hyperparameter tuning on neural networks models. In *26th European Modeling & Simulation Symposium*. (Burdeos, France). 402–409.

[36] Courtney Paquette and Katya Scheinberg. 2020. A stochastic line search method with expected complexity analysis. *SIAM Journal on Optimization* 30, 1 (2020), 349–376.

[37] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics* (1951), 400–407.

[38] Paul J. Sanchez. 2021. datafarming (version 1.4.0): RubyGem. (2021). https://rubygems.org/gems/datafarming

[39] Paul J. Sanchez and Susan M. Sanchez. 2019. Orthogonal second-order space-filling designs with insights from simulation experiments to support test planning. *Quality and Reliability Engineering International* 35, 3 (2019), 854–867. https://doi.org/10.1002/qre.2424 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.2424

[40] Susan M. Sanchez. 2020. Data farming: Methods for the present, opportunities for the future. *ACM Transactions on Modeling and Computer Simulation* 30, 4 (2020), Article 22. 1–30.

[41] S. M. Sanchez and P. J. Sanchez. 2005. Very large fractional factorial and central composite designs. *ACM Transactions on Modeling and Computer Simulation* 15, 4 (2005), 362–377.

[42] Susan M. Sanchez and Paul J. Sanchez. 2020. Robustness revisited: Simulation optimization viewed through a different lens. In *Proceedings of the 2020 Winter Simulation Conference*, Ki-Hwan G. Bae, Ben Feng, Sojung Kim, Sanja

Lazarova-Molnar, Zeyu Zheng, Theresa Roeder, and Renee Thiesing (Eds.). Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ.

[43] Susan M. Sanchez, Paul J. Sanchez, and Wan Hong. 2021. Work smarter, not harder: A tutorial on designing and conducting simulation experiments. In *P 2021 Winter Simulation Conference*, S. Kim, B. Feng, S. Masoud, Z. Zheng, C. Szabo, and M. Loper (Eds.). Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ.

[44] Dustin Schultz. 2015. *High Energy Laser Employment in Self-defense Tactics on Naval Platforms in a Surface Threat Environment*. Master's thesis. Naval Postgraduate School, Monterey, CA.

[45] Sara Shashaani, Fatemeh S Hashemi, and Raghu Pasupathy. 2018. ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free stochastic optimization. *SIAM Journal on Optimization* 28, 4 (2018), 3145–3176.

[46] Stephen C. Upton, Mary L. McDonald, Susan M. Sanchez, and Holly M. Zabinski. 2017. Invoking "ARTeMIS"—the multi-objective hunt for diverse and robust alternative solutions. In *Proceedings of the 2017 Winter Simulation Conference*, W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page (Eds.). Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, 4501–4502.

[47] Hélcio Vieira, S. M. Sanchez, K. H. K. Kienitz, and M. C. N. Belderrain. 2013. Efficient, nearly orthogonal-and-balanced, mixed designs: An effective way to conduct trade-off analyses via simulation. *Journal of Simulation* 7, 4 (2013), 264–275.

[48] Alejandro F. Villaverde, Fabian Fröhlich, Daniel Weindl, Jan Hasenauer, and Julio R. Banga. 2019. Benchmarking optimization methods for parameter estimation in large kinetic models. *Bioinformatics* 35, 5 (2019), 830–838.

[49] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine* 104 (2020), 101822.

[50] Tong Yu and Hong Zhu. 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

[51] Zhi Yuan, Marco A Montes de Oca, Mauro Birattari, and Thomas Stützle. 2012. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence* 6 (2012), 49–75.

Received 22 June 2023; revised 5 March 2024; accepted 16 July 2024