Model-Data Ecosystems: Challenges, Tools, and Trends

Peter J. Haas
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099 U.S.A.
phaas@us.ibm.com

ABSTRACT

In the past few years, research around (big) data management has begun to intertwine with research around predictive modeling and simulation in novel and interesting ways. Driving this trend is an increasing recognition that information contained in real-world data must be combined with information from domain experts, as embodied in simulation models, in order to enable robust decision making under uncertainty. Simulation models of large, complex systems (traffic, biology, population well-being) consume and produce massive amounts of data and compound the challenges of traditional information management. We survey some challenges, mathematical tools, and future directions in the emerging research area of model-data ecosystems. Topics include (i) methods for enabling data-intensive simulation, (ii) simulation and information integration, and (iii) simulation metamodeling for guiding the generation of simulated data and the collection of real-world data.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*decision support*; I.6 [**Simulation and Modeling**]: Simulation Support Systems

General Terms

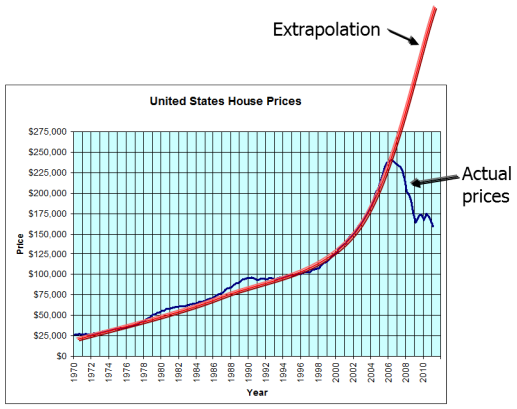
Algorithms, Design

Keywords

Simulation, data assimilation, information integration, decision support

1. INTRODUCTION: DATA IS STILL DEAD

In their *VLDB 2011* paper, "Data is dead…without what-if analytics", Haas et al. [27] point out that, outside of scientific or historical investigations and monitoring-type applications, the essential motivation underlying data processing and analytics is the need to support enterprise decision making under uncertainty. Thus the ultimate goal is to support deep predictive analytics that incorporate domain expertise in order to robustly predict the future consequences of decisions made today. From this perspective, data by itself is indeed "dead", reflecting the past state of the world. Descriptive analytics—such as simple querying, OLAP, data mining, machine learning, and time-series analysis—find important patterns and relationships in existing data, leading to insights about the real world as it currently stands. A "shallow" predictive approach that simply extrapolates current patterns into the future, however, can lead to very brittle predictions and subsequent bad decisions because it does not account for the fact that the mechanisms that generated the existing data can change. Figure 1 illustrates this point. A simple time series model was fit to median U.S. housing prices from 1970 to 2006 and then extrapolated to 2011. As can be seen, the resulting prediction failed spectacularly because it ignored expert information from economists, financial analysts, behavioral scientists, and others that might have helped in modeling the housing-price collapse that began in 2006. Thus data must be supplemented by models that embody expert knowledge about the constituent parts of systems and the way they behave and interact. For systems characterized by uncertainty, these models often take the form of stochastic simulations.



Extrapolation of 1970-2006
median U.S. housing prices

Figure 1: The dangers of extrapolation

Eric Bonabeau, the author of *Swarm Intelligence*, makes a similar point in one of his blogs [9]:

> There is no doubt that the more *information* is used in building a model, the more accurate the model is likely to be. However, the notion that quantitative, nu-

PODS'14, June 22–27, 2014, Snowbird, UT, USA.

ACM 978-1-4503-2375-8/14/06 ...$15.00.
http://dx.doi.org/10.1145/2594538.2594562.

merical data are the *only* type of information needed to build an accurate model is flawed. In fact, I believe that the typical business obsession with numeric data can do more damage than good.

He then asserts the need for fundamental system information possessed by domain experts and gives as an example the problem of reducing traffic jams. A data-centric approach would collect large amounts of data about current traffic speeds, volumes, and delays, and attempt to discover correlations between time of day and average speed, between number of cars and delay lengths, and so on. This approach ignores crucial information possessed by traffic experts, such as that we slow down at certain rates when someone appears in front of us, that we accelerate to a driver-dependent "comfortable" speed when the road is clear, that we may switch lanes if they are open, and so on. "This kind of information is critical because it is at the heart of what creates traffic. And yet, typical data-driven approaches would have no way whatsoever of including this domain knowledge along with the numerical data." Bonabeau then shows how simple agent-based simulations that incorporate such behavior can accurately imitate traffic jams observed in the real world. In this context, data is key to parametrizing and calibrating such models.

Perhaps because of their increasing awareness of the need for interplay between models and data within a coherent ecosystem, researchers in the fields of information management and system simulation have interacted more and more over the past several years, and this trend will most likely continue. In a striking example of this confluence of interests, the *2014 Winter Simulation Conference (WSC)*—the premier venue of the stochastic simulation community—has chosen for its theme "exploring big data through simulation". Research on model-data ecosystems, however, is clearly in its infancy. Indeed, even the Conference Chair of WSC 2014 has expressed uncertainty about the precise meaning of the theme.

We therefore provide a survey of research pertaining to the emerging model-data ecosystem. Each topic will be treated via specific examples, presented in a tutorial style. The selection of topics is admittedly idiosyncratic, reflecting the author's interests and experience as a member of both the database and simulation communities. The goal is both to spark interest in some of these topics and to indicate some system prototypes and mathematical tools—some of which are perhaps more familiar to the modeling community than to the database community—that are being used to attack some of the new research questions. Overall, we distinguish three main topics:

1. *Data-intensive simulation* Modern simulation models can consume and produce massive amounts of data, which creates opportunities to improve simulation performance by deploying data-management technologies within simulation environments. These include (i) incorporating stochastic simulation functionality into database systems, (ii) developing methods for massive scale time series transformations between models using MapReduce and related technologies, (iii) extending techniques for query optimization to the setting of simulation-run optimization, and (iv) querying simulated data as part of the simulation process.

2. *Simulation and information integration* An interesting line of thought views simulation, especially agent-based simulation (ABS), as a tool for combining disparate real-world data. A related topic is "data assimilation", which is concerned with fusing simulated and real-world sensor data to get a clearer picture of an evolving real-world phenomenon such as a forest fire.

3. *Simulation metamodeling* Metamodeling techniques provide a powerful set of tools both for controlling the amount of simulated data that is generated and for guiding the collection of real-world data.

In the following sections we will cover each of these topics in turn.

2. DATA-INTENSIVE SIMULATION

Traditionally, a simulation analysis would start with a small to medium set of input data and produce a medium to "large" set of output data. Both the input data and output data would themselves be the result of a data reduction process. Typical input data, for example, might comprise a so-called "reproduction number" for use in an epidemic simulation or some mean and variance parameters of a lognormal distribution for use in a financial simulation; such inputs would typically be the result of a statistical estimation procedure. This situation has changed dramatically with the introduction of agent-based simulation—see, e.g., [2, 7, 39]. ABS is an approach to modeling systems comprising individual, autonomous, interacting "agents". With roots going back at least to the 1970's [48], ABS has surged in popularity in recent years as computational power has grown. An ABS often involves massive numbers of agents, e.g., in simulations of regional or national populations at the individual level. Thus the input data requirements can be huge. The output of the simulation, which can comprise a time series of snapshots of the state of the agent population, can also be massive. Even when ABS is not used *per se*, the scale, complexity, and granularity of simulations is growing rapidly, commensurate with increases in both computing power and the amount of available data. Running a modern simulation therefore requires managing large amounts of data, and there have been a number of attempts to leverage the expertise of the information-management community when designing simulation platforms.

2.1 Simulation in the database

One line of research has attempted to incorporate stochastic simulation functionality into database systems, with a goal of avoiding the need to reduce, extract, and load the data into a separate simulation engine and then later integrate the simulation results back into the database. Examples of this type of system are given by the *Monte Carlo Database System (MCDB)* and *SimSQL*. The MCDB system [33] allows an analyst to attach arbitrary stochastic models to a relational database. In more detail, the analyst can specify, in addition to the ordinary tables in the database, "stochastic" tables that contain "uncertain" data. Such data are not represented by specific data values, but rather by stochastic models that describe the probability distribution over possible values. The models are implemented as user- and system-defined libraries of external C++ programs called *Variable Generation functions*, or VG functions for short. A call to a VG function generates a realization of uncertain data values in the form of a pseudorandom sample from the underlying probability distribution. The sample can correspond to the value of a single data element or to a set of correlated elements residing in various row and column positions. The possible actions of a VG function range from simple generation of a sample from a normal distribution, to executing a backward random walk starting at a given current price in order to estimate missing prior prices, to simulating a sequence of stock prices in order to return a sample of the value of a stock option one week from now. As another example, a customer's random demand for an item, given its price, might be computed by fitting a parametric global demand model based on data from all customers, and then computing a customized demand distribution for each customer using the customer's individual pur-

chase history together with Bayes' Theorem. Then one can ask queries such as "how would the revenue from East Coast customers under thirty years old have been affected by a 5% price increase?".

In MCDB, the VG functions are parametrized on the current state of the non-random tables (e.g., tables of historical sales data or delivery times). As a very simple example, consider the following specification of a random table of blood pressure data:

```
CREATE TABLE SBP_DATA(PID, GENDER, SBP) AS
 FOR EACH p in PATIENTS
  WITH SBP AS Normal (
   SELECT s.MEAN, s.STD
   FROM SPB_PARAM s)
  SELECT p.PID, p.GENDER, b.VALUE
  FROM SBP b
```

A realization of `SBP_DATA` is generated by looping over the set of patients and using the `Normal` VG function to generate a row for each patient. These rows are effectively `UNION`ed to create the realization of `SBP_DATA`. The `FOR EACH` clause specifies this outer loop. The `Normal` VG function simple generates a sample from the normal distribution. This function is parametrized with a mean and standard deviation, which are obtained via a SELECT query over the (single row, two column) `SBP_PARAM` table; in general a VG function can be parametrized using a general SQL query over the set of all non-random relations in the database.

Generating a sample of each uncertain data value creates a *database instance*, i.e., a realization of an ordinary database. Running an SQL query over the database instance generates a sample from the query-result distribution. Iteration of this process yields a collection of samples from this distribution that can then be used to estimate distribution features of interest such as moments and quantiles. To ensure acceptable performance, MCDB employs query processing techniques that execute a query plan only once, processing "tuple bundles" rather than ordinary tuples. A tuple bundle encapsulates the instantiations of a tuple over a set of Monte Carlo iterations. MCDB runs on a parallel relational database platform, and so can exploit parallel database technology for scalability. Subsequent work [5, 42] demonstrates how MCDB can be extended to deal with risk analysis (by efficiently estimating extreme quantiles) and with threshold queries of the form "Which regions will see more than a 2% decline in sales with at least 50% probability?".

SimSQL [11] is a re-implementation and extension of MCDB. SimSQL allows data in stochastic database tables to be used to parametrize the VG functions that generate the data in other stochastic database tables. Moreover, SimSQL allows both versioning and recursive definitions of stochastic database tables. For example, data in stochastic table `A` can be used to parametrize the stochastic generation of table `B`, which in turn can be used to parametrize the stochastic generation of a second version of table `A`, and so on. Whereas MCDB merely allowed generation of sample realizations of a stochastic database D—in other words, a static database-valued random variable—the foregoing extensions enable SimSQL to generate realizations of a database-valued Markov chain $D[0], D[1], D[2], \ldots$. That is, the stochastic mechanism that generates a realization of the ith database state $D[i]$ may explicitly depend on the prior state $D[i-1]$. As with MCDB, queries are expressed in SQL; unlike MCDB, SimSQL executes queries using the Hadoop [4] MapReduce implementation in order to scale to massive data.

Besides being well suited to scalable Bayesian machine learning (see [11]), SimSQL can also be used to implement massive stochastic ABS models inside the database. The idea is to build on work by Wang et al. [55], who observed that a step in an agent-based simulation can be viewed as a self-join. That is, the data in each row of a table represent the internal state of an agent, so the self-join step allows agents to interact with other agents. A key observation is that agents typically interact only with a relatively small group of "nearby" agents. Thus (with a little care) the join can be parallelized among groups of agents, and well known parallel database technology can be applied to achieve good performance. The work in [55] applies primarily to deterministic simulations; SimSQL can potentially be used to extend those ideas to stochastic simulations.

2.2 Data Harmonization at Scale

The need for transforming large datasets is certainly not new. The scientific community, especially, has long been concerned with transforming spatio-temporal data as part of a scientific workflow, both to combine disparate datasets and to conform datasets to the formats expected by analysis programs. Over the past few years, however, demand for efficient data transformations at scale has increased even further, driven by the rise of composite simulation modeling.

Decision-makers increasingly need to bring together multiple models across a broad range of disciplines to guide investment and policy decisions around highly complex issues such as population well-being; see, e.g., [23, 32] in the setting of food, climate, and health. Consequently, there have been a number of efforts to develop composite modeling tools that can leverage both new and existing models developed by domain experts. Examples include CIShell [10], Open Services Gateway initiatives (OSGi) framework, the High Level Architecture for U.S. Department of Defense simulations [35], the DEVS discrete-event simulation framework [54], and targeted simulation frameworks such as the STEM epidemiological model [18] and the Community Climate System Model [13], in which component models are written in a specified language (Java and Fortran 90) and compiled together to create the simulation program. All of these frameworks require major re-coding of existing models and/or strict enforcement of a common platform, API, or communication protocol—a nearly impossible task when dealing with experts in vastly different domains.

In response to these issues, systems such as the IBM Splash prototype (Smarter Planet Platform for the Analysis and Simulation of Health) [26, 28, 53] have attempted to synthesize simulation and data-integration techniques, permitting loose coupling of models via data exchange; that is, models communicate by reading and writing datasets. When model and data contributors initially register their models and datasets with Splash, they provide metadata that enables drag-and-drop composite-model creation, automatic detection of data mismatches between upstream "source" and downstream "target" models, and graphical tools for specifying needed data transformations, which are then compiled into runtime code. For a stochastic composite model, data transformations must be performed at every Monte Carlo repetition, so that efficiency a is major concern.

The database community has lent its expertise to improving data-transformation technology in several ways. For example, Howe and Maier [31] provide an algebra of "gridfields" to allow for efficient optimization and query processing of gridded data in a relational database, especially when the grids are irregular. A *grid*, in their terminology, is a collection of heterogeneous abstract *cells* of various dimensions. A grid also has an incidence relation $\preceq$ between cells, where $x \preceq y$ means that either $x = y$ or $\dim(x) < \dim(y)$ and x "touches" (i.e., is adjacent to) y. For example, cell x might correspond to a line segment and cell y might correspond to a square, so that $x \preceq y$ if x coincides with a side of the square. A variety of set-like operations can be defined on grids. A *gridfield* results from binding data to a grid by specifying, for each dimension k, a function f_k that operates on cells of dimension k and returns a

data value of some type τ_k. A variety of operations can be defined on gridfields, with the most important operation in the current context being "regrid". The regrid operator maps a source gridfield's cells onto a target gridfield's cells via a many-to-one assignment function and then aggregates the data values bound to the mapped cells via an aggregation function. The authors show, for example that certain "restriction" operations—which are analogous to standard relational selection operations—-can commute with the regrid operator, creating opportunities for optimization. This technology was originally applied to the CORIE system, which supports simulations of the Columbia River Estuary and other coastal regions.

As another example, the Splash system, discussed earlier, uses Hadoop for data transformations between models. Hadoop is used both to execute *schema alignment* and *time alignment* transformations on massive time series with many time ticks and large amounts of data per tick. The former type of transformation typically handles "format" discrepancies between source-model outputs and target-model inputs at any given point of simulated time, whereas time alignment deals with the orthogonal problem of timescale discrepancies between models. To specify schema transformations, Splash uses Clio++, an extension of the Clio schema mapping tool [24] to allow users to graphically define a schema mapping. For time-alignment transformations, the time aligner tool determines the class of time alignment needed—e.g., aggregation if the target model has coarser time granularity than the source model or interpolation if the target has finer granularity—and provides a GUI that lets the user specify an appropriate alignment method from a menu. Graphical specifications of data transformations are then automatically compiled into Hadoop runtime code.

Some time alignment operations are non-trivial to implement efficiently in a MapReduce setting. Consider interpolation, for example, where the input is a time series $S = \langle (s_0, d_0), \ldots, (s_m, d_m) \rangle$ for some $m \geq 0$; here s_i is the time of the ith observation and d_i is the associated data observed at time s_i. Each d_i can be viewed as a k-tuple for some $k \geq 1$. The output is a target dataset $T = \langle (t_0, \tilde{d_0}), \ldots, (t_n, \tilde{d_n}) \rangle$. To exploit parallelization, Splash computes *windows* of the form $W = \langle (s_j, d_j), (s_{j+1}, d_{j+1}) \rangle$. Each window is used to compute data at target points $\{ t_i : s_j \leq t_i < s_{j+1} \}$. The windows can be processed in parallel and then the target time series can be assembled via a parallel sort. A novel challenge arises when computing a natural cubic spline interpolation, one of the most common interpolations used in practice. The interpolation formula is

$$\tilde{d_i} = \frac{\sigma_j}{6h_j}(s_{j+1} - t_i)^3 + \frac{\sigma_{j+1}}{6h_j}(t_i - s_j)^3 + \left(\frac{d_{j+1}}{h_j} - \frac{\sigma_{j+1}h_j}{6}\right)(t_i - s_j) + \left(\frac{d_j}{h_j} - \frac{\sigma_j h_j}{6}\right)(s_{j+1} - t_i),$$

where $h_j = s_{j+1} - s_j$ and $\sigma_0, \sigma_1, \ldots, \sigma_m$ are *spline constants* that depend on the entire input dataset and are computed as the solution to linear equation system of the form $Ax = b$, where A is an $(m-1) \times (m-1)$ tridiagonal matrix. For massive time series with fine granularity, m can be huge, and the matrix A can contain millions of rows and millions of columns. Virtually all known methods for solving such massive systems do not translate well to a MapReduce environment, because massive amounts of data shuffling are required.

The technique used in [28] is to transform the problem of solving the tridiagonal linear system into the problem of choosing x to minimize $L(x) = \|Ax - b\|^2$—where $\|z\|$ denotes the euclidean norm—and then to apply a *distributed stochastic gradient descent* (DSGD) approach [21]. Ordinary stochastic gradient descent (SGD), minimizes L by taking downhill steps, using "quick and dirty" approximations of the true gradient. In more detail, we first write $L(x) = \sum_{i=1}^{m-1} L_i(x)$, where $L_i(x) = (A_{i\cdot}x - b_i)^2$ and $A_{i\cdot}$ denotes the ith row of A. The SGD algorithm starts with an initial guess $x^{(0)}$ for x, then picks a row I at random from $\{1, 2, \ldots, m-1\}$, computes the gradient component $\nabla L_I(x^{(0)})$, then approximates the overall gradient $\nabla L(x^{(0)})) = \sum_{i=0}^{m} \nabla L_i(x^{(0)})$ by $Y_0 = m\nabla L_I(x^{(0)})$, and finally updates the solution by setting $x^{(1)} = x^{(0)} - \varepsilon_0 Y_0$. Such downhill steps are iterated using a carefully chosen sequence $\{\varepsilon_n\}_{n \geq 0}$ of step sizes; for step sizes of the form $\varepsilon_n = n^{-\alpha}$, SGD is provably convergent under mild conditions, provided that $1 \leq \alpha < 2$.

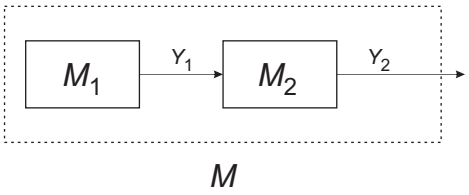


Figure 2: A simple composite simulation model

The SGD approach has long been known to work well in sequential settings [43] but it has not been obvious how to distribute the algorithm across a cluster. This problem was addressed in [21], originally in the context of matrix completion problems arising in recommendation systems. The idea is to partition the data into *strata*; the strata are carefully defined so that execution of SGD within a stratum can be parallelized. The DGSD algorithm runs within a chosen stratum for a period of time, then switches to another stratum, and so on. For the cubic spline problem, the first stratum S_1 comprises the data in rows $1, 4, 7, \ldots$ of the linear system. If row $i = 1$ is selected at random in SGD, the tridiagonal structure of A guarantees that the resulting update to x will only involve entries x_1 and x_2. Similarly, an update to row $i = 4$ will only involve entries x_3, x_4, and x_5. Thus rows 1 and 4 can be sampled in either order, or in parallel, and indeed the data in S_1 can be partitioned among the processing nodes and processed in parallel by the SGD algorithm. Similarly, SGD can be run in parallel over data in stratum $S_2 = \{2, 5, 8, \ldots\}$ and in $S_3 = \{3, 6, 9, \ldots\}$. Observe that when running in a given stratum S_i, SGD is minimizing the "wrong" function because the data in the other strata are being ignored. It is shown in [21], however, that if the process switches randomly from one stratum to another according to a "regenerative" process, and if equal time is spent in each stratum in the long run, then the algorithm will converge to the overall solution with probability 1. Moreover, the amount of data that needs to be shuffled is negligible, which results in superior performance on MapReduce platforms. Recent experiments indicate that the DSGD idea also leads to best-of-breed matrix completion algorithms on a variety of architectures [40].

2.3 Optimizing Simulation Runs

Optimizing the efficiency of simulation execution is another challenge where the simulation community can benefit from the experience of the database community. In particular, composite-modeling platforms such as Splash need to execute queries in order to harmonize data between models during a simulation run, so that the problem of simulation-experiment optimization subsumes the problem of query optimization. The challenges go further, however, to include issues such as how best to move data to models, or vice versa, in a distributed execution environment, how to schedule execution of components to minimize the overall run time, and so on.

We briefly discuss an example, taken from [25], of an optimization problem specific to stochastic composite simulation but with links to classical query optimization technology. To motivate the problem, consider a composite model M comprising two component models M_1 and M_2 in series, as shown in Figure 2. An execution of M proceeds by first executing M_1, which produces a random output Y_1 that is written to disk. Then M_2 is executed, taking Y_1 as input (after appropriate transformation) and generating a final output Y_2, where Y_2 is distributed according to a conditional cumulative distribution function $F_2(\cdot|Y_1)$. For example, M_1 might be a demand model that generates a sequence Y_1 of customer arrival times. The data in Y_1 might then fed into a queuing model M_2, which in turn produces an output Y_2, which might correspond to the average waiting time of the first 100 customers. To generate independent and identically distributed (i.i.d.) samples $Y_{2;1}, Y_{2;2}, \ldots, Y_{2;n}$ from the distribution of Y_2, one could execute the composite model n times, thereby executing M_1 and M_2 a total of n times each. Suppose, however, that M_1 is in fact deterministic, so that the same output Y_1 is produced every time the model is executed. Rather than repeatedly executing M_1. It is clearly more efficient to cache the output Y_1 of M_1 during the first execution of the composite model M and then, for each of the remaining $n-1$ executions, to read Y_1 from disk instead of re-executing M_1. If the cost of executing M_1 is large relative to the cost of executing M_2, then the cost savings can be significant. The general question, then, is how to optimally reuse results for a general composite model in which each component model might be stochastic.

The result-caching (RC) technique for the case of two stochastic models in series in considered in [25]. Here the goal of the simulation is to estimate $\theta = E[Y_2]$, the expected value of the (real valued) output from M_2. For n simulation replications of M_2, only $m_n = \lceil \alpha n \rceil$ replications of M_1 are executed, where $\alpha \in (0,1]$ is called the *replication fraction*.[1] We write the output of M_1 to disk after each of the first m_n simulation replications and then repeatedly cycle through these outputs in a fixed order to obtain inputs to M_2. Thus each M_1 output is used in approximately n/m_n executions of M_2. The deterministic cycling scheme produces a stratified sample of the outputs of M_1 and helps minimize estimator variance. Finally, θ is estimated as $\theta_n = (1/n)\sum_{l=0}^{n-1} Y_{2;l}$.

To precisely formulate the notion of "increasing the efficiency" of a simulation run, suppose that we are given a large but finite computing budget c. Denote by C_n the cost of generating n outputs from M_2 under the RC strategy; this cost comprises the cost of creating, transforming, and storing m_n outputs from M_1 plus the cost of creating n outputs from M_2. Under a budget c, the number of M_2 outputs that can be generated is $N(c) = \sup\{n \geq 0 : C_n \leq c\}$, resulting in the estimate $U(c) = \theta_{N(c)}$. Denote by c_1 and c_2 the expected computation costs for the first run of M_1 and M_2, respectively (so that the costs of transforming and storing the output from M_1 are included). Also denote by V_1 the variance of an output from M_2 and by V_2 the covariance of two outputs from M_2 when they share a common input from M_1; assume that $V_2 \geq 0$, as is usually the case in practice. Finally, set $r_\alpha = \lfloor 1/\alpha \rfloor$. It can be shown that $U(c) \to \theta$ with probability 1 and $c^{1/2}[U(c) - \theta] \Rightarrow \sqrt{g(\alpha)}N(0,1)$ as $c \to \infty$, where "$\Rightarrow$" denotes convergence in distribution, $N(0,1)$ denotes a standard (mean 0, variance 1) normal random variable, and

$$g(\alpha) = (\alpha c_1 + c_2)\Big(V_1 + [2r_\alpha - \alpha r_\alpha(r_\alpha + 1)]V_2\Big).$$

[1]Throughout, $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the largest integer less than or equal to x and the smallest integer greater than or equal to x.

Thus $U(c)$ is a strongly consistent estimator of θ and, for a large budget c, is approximately normally distributed with mean θ and variance $g(\alpha)$. Following [19, 22], one can then define (asymptotic) efficiency as $1/g(\alpha)$, so that maximizing efficiency is equivalent to minimizing the variance of a budget-constrained simulation-based estimator. It can be seen that $g(\alpha)$ is the amortized total cost to produce an output from M_2 times the variance of such an output. This product-form balancing of simulation cost and variance in order to obtain an overall measure of efficiency was originally proposed by Hammersley and Handscomb in 1964 [29]; consideration of the asymptotic behavior of budget-constrained simulations, as above, provide a rigorous justification for this type of efficiency measure.

To gain insight into the nature of the optimal solution for the two-model problem, approximate r_α by $1/\alpha$ and write $g(\alpha) \approx \tilde{g}(\alpha) \stackrel{\text{def}}{=} (\alpha c_1 + c_2)\big(V_1 + (\alpha^{-1} - 1)V_2\big)$. It is then easy to verify that $\tilde{g}(\alpha)$ is maximized by setting $\alpha = \alpha^*$, where

$$\alpha^* = \left(\frac{c_2/c_1}{(V_1/V_2) - 1}\right)^{1/2}.$$

(Truncate α^* at $1/n$ or 1 as needed to ensure a feasible solution. Note that $V_1/V_2 \geq 1$ by the Cauchy-Schwarz inequality.) If c_1 is large relative to c_2, so that M_1 is relatively expensive to execute, then α^* is small, and it is optimal to execute M_1 a relatively small number of times. If M_2 is relatively insensitive to the input from M_1, so that most of the variability arises from randomness within M_2, then $V_2 \ll V_1$, and again we simulate M_1 only a small number of times. In the extreme case where $V_2 = 0$, we simulate M_1 only once, thereby recapturing the scenario in which M_1 is deterministic. If, on the other hand, M_2 is a deterministic function of its inputs and M_1 is stochastic, then $V_1 = V_2$ and it is optimal to run n replications of M_1. In this case, M_2 is merely a transformer of the output of M_1. Overall, it is clear that, depending on the values of c_1/c_2 and V_1/V_2, arbitrarily large efficiency improvements are possible in principle.

A key issue is how to estimate the statistics $\mathscr{S} = (c_1, c_2, V_1, V_2)$. Note that there is some tolerance for error: inaccuracies in the estimates of $\mathscr{S}$ might result in slightly suboptimal simulation performance but correctness is not an issue. Indeed, the foregoing results show that the simulation estimates are asymptotically valid for any value of α. Also note that a composite modeling system such as Splash is oriented toward re-use of models, and important performance characteristics of a model can be stored as part of the model's metadata. Thus the cost of executing pilot runs to estimate the statistics in $\mathscr{S}$ can be amortized over multiple model executions. Moreover, as the component models are used in production runs, their behavior can be observed and used to continually refine the statistics in $\mathscr{S}$, and hence to continually improve performance. The issues here are analogous to the issues encountered in estimating catalog statistics for a relational database system, and thus techniques from traditional query optimization can potentially be leveraged in the simulation setting and combined with metamodeling ideas as in Section 4 below and perhaps ideas from machine learning; see [25] for further discussion.

2.4 Querying Data During a Simulation

As we have seen with systems such as MCDB and SimSQL, one way to leverage information-management technology in data-intensive simulation is to incorporate simulation functionality into a database system. Another approach—as embodied in the Indemics simulation model for large scale epidemics [6]—is to divide the simulation work between a high-performance cluster (HPC) that performs compute-intensive tasks and a relational database engine that performs data-intensive tasks.

In more detail, Indemics uses a network model of disease transmission, where nodes represent individuals and edges represent social contacts between individuals. The nodes have attributes representing the health and behavioral state of an individual, along with static demographic information, and the edges have attributes that specify, e.g., contact duration and type. The model also comprises transition functions that modify nodes and/or edges, and hence specify changes in disease progression and behavioral status (e.g., fear level), as well as changes in social interactions (formation of new edges due to new contacts, deletion of edges due to quarantine, and so on). The HPC updates the state of the network in between observation times. At an observation time, the experimenter can issue SQL queries to assess the state of the network model. Such queries select subsets of individuals and run aggregation queries on each subset to summarize the state of the subpopulations (e.g., percent infected). Queries can also be used compute values of performance measures that are to be optimized (e.g., number of infected cases or economic damage). Finally, SQL queries can be used to specify complex interventions by specifying subsets of individuals together with the actions to be performed on each subset. An action on a subset S corresponds to modifying the states of the nodes corresponding to the individuals in S and/or the edges incident on the individuals in S. Algorithm 1, adapted from [6], indicates how an intervention strategy can be specified using SQL. When executed by the experimenter after pausing the simulation, this type of intervention represents an interactive extension to traditional "partially observed Markov decision processes".

Algorithm 1 Vaccinate preschoolers if more than 1% are sick

```
CREATE TABLE Preschool(pid) AS
  (SELECT pid FROM Person WHERE 0 ≤ age ≤ 4);
/* Based on demographic data */
DEFINE nPreschool AS (SELECT COUNT(pid) FROM Preschool);
for day = 1 to 300 do
    /* Based on demographic and disease dynamic data */
    WITH InfectedPreschool (pid) AS
     (SELECT pid FROM Preschool, InfectedPerson
      WHERE Preschool.pid = InfectedPerson.pid);
    DEFINE nInfectedPreschool AS
     (SELECT COUNT(pid) FROM InfectedPreschool);
    if nInfectedPreschool > 1% × nPreschool then
        Apply vaccines to SELECT( pid FROM Preschool);
        /* Intervention subpopulation and action */
    end if
end for
```

This division of labor between an HPC and an RDBMS yields a highly extensible and flexible system for specifying disease transmission models together with complicated intervention policies. This approach can potentially be applied to other large scale agent-based simulations.

We briefly mention another type of querying inside simulations, specifically, inside parallel discrete-event simulations of agent-based systems. Such simulations are studied in [52], for a simulation platform called PDES-MAS. In such simulations, parallel "agent logical processes" (ALPs) simulate the simultaneous behavior of massive numbers of agents. Each agent operates in a repeating cycle of "sense-think-response". A key part of the "sense" stage is discovering nearby agents via an instantaneous *range* query, e.g., "find all agents who are, right now, within one mile and who are over 25 years old". In the PDES-MAS distributed architecture, a set of "communication logical processes" (CLPs) maintains, in a distributed manner, a collection of "shared-state variables" (SSVs) that describe the state of the environment as well as the externally viewable characteristics of the agents such as physical location. CLPs in fact maintain a history of SSV values over time. In the PDES-MAS system, LPs communicate through ports; the CLPs are arranged in a treelike structure with leaves corresponding to ALPs, in such a manner as to balance accesses to the SSVs. The tree of CLPs is dynamic, with possible reconfiguration of the tree structure and migration of SSVs and/or ALPs in a continual attempt to move SSVs closer to the ALPs that are accessing them. Because the ALPs may progress through simulated time at different rates, answering range queries correctly becomes extremely challenging. The authors in [52] provide some initial range-query algorithms and test them empirically; there is still a need for theoretical analysis of such algorithms.

3. INFORMATION INTEGRATION

The discussion so far has concerned the combination of information-management and simulation methods to create simulation tools that can handle massive data. Another set of interesting ideas concerns information integration, both the integration of heterogeneous real-world datasets and the integration of real-world and simulated data. We discuss these two aspects below.

3.1 Simulation as an integration tool

In his keynote speech at WSC 2013 [8], Eric Bonabeau postulated that agent-based simulations can be viewed as a powerful tool for data integration and an essential means for making sense of big data. As one concrete example, he described issues that arise in marketing. Typically the available datasets are quite disparate in nature and measured at a variety of granularities. These include data about

1. Individual consumer behaviors (purchasing, use of product, communicating about product)
2. Aggregate customer profiles (drivers, awareness, perceptions)
3. Network data (connections, relationships, influence)
4. Touch points (channel affinities, media impact, reach and frequency)
5. Decisionmaking (rationality, emotion, social norms)

The claim is that big data investments alone will not solve the forecasting problem, because such investments merely provide non-overlapping perspectives on individuals from disparate data sources characterized by varying levels of aggregation. To address this problem, an ABS approach can be used to simulate synthetic personas created from these heterogeneous data sources. The key is then to *calibrate* the model using statistical and machine learning techniques in order to approximately match existing datasets. This integrated model can then be used to understand and predict the effect of different types of touch points, perception changes, social network influence, stages of the purchase "journey", and thus to forecast volatility, risk, and reward associated with marketing strategies. In more detail, a brand tracker or survey data can provide customer properties, media and sales data describe marketing effectiveness, industry or product reports cover the product or offer, and social tracking data describe word-of-mouth events and behaviors. The ABS model brings together these four disparate datasets in an integrated way and yields rich insights into consumer behavior that go far beyond mere sales.

Research on calibrating agent-based models is still at an early stage. Many of the existing quantitative techniques developed so far have come from the econometrics community, which has enthusiastically embraced the use of agent-based models. The traditional

approach to estimating parameters is the method of *maximum likelihood* [37]. As a simple example, consider data $X = (X_1, X_2, \ldots, X_n)$ representing a sample of i.i.d. draws from the exponential density function $f(x;\theta) = \theta e^{-\theta x}$ for $x \geq 0$. The likelihood of seeing the given data points is $L(\theta;X) = \prod_{i-1}^{n} f(X_i;\theta) = \theta^n e^{-\sum_i X_i}$. The maximum likelihood estimate (MLE) $\hat{\theta}_n$ of θ is obtained by choosing θ to maximize $L(\theta;X)$ (or, equivalently, the logarithm of the likelihood L). A simple calculation yields $\hat{\theta}_n = 1/\bar{X}_n$, where $\bar{X}_n$ is the average of the X_i's. Although MLEs have many desirable properties, the output of an ABS is usually highly nonlinear and complex, so that the likelihood can only be obtained in rare cases; see [1] for an example.

Because of these difficulties, attempts at quantitative calibration have primarily relied on adaptations of the *method of moments* (MM). For the simple exponential example, it is known that the mean of an exponential random variable is given by $E[X] = 1/\theta$. The method of moments proceeds by replacing $E[X]$ by its empirical counterpart $\bar{X}_n$ and solving for θ, which in this case yields the MLE estimator (although in general the MM and MLE estimators do not coincide). For, e.g., a normal distribution, two equations in two unknowns would be used, equating the first two moments to the sample mean and sample variance. More generally, the procedure centers on a vector of observed statistics $Y = (Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)})$ and, for $\theta = (\theta_1, \theta_2, \ldots, \theta_m)$, solves the system $\bar{Y}_n - m(\theta) = 0$, where $Y_1, Y_2, \ldots, Y_n$ are i.i.d. samples of Y and $m(\theta) = E[Y|\theta]$.

The extension of this method to ABS calibration is usually called the *method of simulated moments* (MSM), and is usually attributed to McFadden [41]. In this more general setting, the observations $Y_1, Y_2, \ldots, Y_n$ may represent a stationary ergodic sequence; i.i.d. observations are a special case. Moreover, $m(\theta)$, which is usually too complex to be calculated analytically, is approximated by a simulation-based estimate $\hat{m}(\theta)$, typically obtained by averaging i.i.d. samples of Y from simulation runs having parameter values equal to θ. Finally, the problem of solving $G_n = \bar{Y}_n - \hat{m}(\theta) = 0$ is usually relaxed to the problem of minimizing the generalized distance $J(\theta) = G_n^\top W G_n$, where W is chosen to boost statistical efficiency; see [30] for some relevant theory. Typically—see, e.g., [20]—W is chosen to be an estimate of the inverse of the variance-covariance matrix of the random vector G_n.

The main difficulty encountered in calibration of this sort is that $\hat{m}(\theta)$ is usually expensive to compute for even a single value of theta, so that the stochastic optimization problem of minimizing $J(\theta)$ is highly cost intensive. Recently, researchers have begun to apply some techniques from the field of simulation-based optimization to develop workable algorithms. For example, Fabretti [17] uses heuristic optimization methods, such as Nelder-Mead and genetic algorithms, to try and quickly locate the optimal parameter value. While this approach is a vast improvement over random sampling of θ values, the computational requirements can still be high. An alternative approach in [45] carefully uses *design of experiment* (DOE) techniques—in particular, a nearly-orthogonal Latin hypercube design (see Section 4.2 below)—to select representative values of θ to simulate. The method then uses a flexible surface-fitting technique called "kriging" to approximate the function $\hat{m}(\theta)$, and hence $J(\theta)$. This approximated function (also called a *simulation metamodel*) is then minimized to find the desired calibrated values of θ; see Section 4 below for a discussion of metamodeling. Both of these techniques implicitly assume that the simulation is deterministic in nature; other methods that explicitly consider stochastic model response are potentially applicable here. For example, the kriging method used in [45] could potentially be replaced by stochastic kriging and extensions—see, e.g., [44]—which incorporate simulation variability into the fitting algorithm.

Overall, there are ample research opportunities around the use of simulation as a data integration tool, especially with respect to model calibration. As indicated above, algorithm development is still at an early, largely ad hoc stage. Another interesting question is how to extend existing approaches, which calibrate against a small number of population summary statistics, to calibrate at a finer granularity. Such fine-grained calibration might have the potential for avoiding situations where multiple calibrations are all deemed acceptable but lead to very different predictions [51]. Principled methods for avoiding overfitting during the calibration process are also of great interest, since simulation-model calibration is known to be vulnerable to such issues [36, p. 266]. One advantage of the MSM method is that regularization terms can potentially be incorporated into the objective function J to avoid overfitting.

3.2 Combining real and simulated data

Besides integrating disparate real-world datasets, there is an increasing interest in combining simulated and real world data in order to more deeply integrate domain expertise into data analysis. One method receiving attention is *data assimilation* via *particle filtering*. As an example, the work in [56, 57] concerns the problem of monitoring a wildfire. Domain experts have developed simulation models that capture the probabilistic mechanism by which a fire spreads over terrain. During an actual fire, real-world temperature data from the affected region is available as a stream of time-varying readings from a set of sensors. Particle filtering can be used to combine sensor readings with simulated data to yield more accurate estimates of the fire status than could be obtained from either data source alone.

We give a brief introduction to particle filtering, following [16], and then indicate how particle filtering can be used for data assimilation. Particle filtering (also called "bootstrap" filtering) algorithms are a subclass of *sequential Monte Carlo* methods, which in turn are a subclass of *importance sampling* (IS) methods. We discuss each class of methods in turn. Suppose that, for some $n \geq 1$, our goal is to obtain a Monte Carlo approximation of a probability density $\pi_n(x_{1:n}) = \gamma_n(x_{1:n})/Z_n$ on $\mathscr{X}^n$, where $\mathscr{X}$ denotes the common state space of the x_i's and $z_{i:j}$ denotes the vector $(z_i, z_{i+1}, \ldots, z_j)$. Here γ_n is an unnormalized probability density and $Z_n = \int \gamma_n(x_{1:n})\, dx_{1:n}$ is a normalizing constant. A standard Monte Carlo approach draws N independent samples $\{X_{1:n}^i\}_{1 \leq i \leq N}$ from π_n and then approximates π_n by

$$\hat{\pi}_n(x_{1:n}) = (1/N) \sum_{i=1}^{N} \delta_{X_{1:n}^i}(x_{1:n}),$$

where $\delta_{x_0}(x)$ denotes the Dirac density function with unit probability mass at x_0. An expected value $\int g(x_{1:n})\pi_n(x_{1:n})\, dx_{1:n}$ can then be approximated by $\int g(x_{1:n})\hat{\pi}_n(x_{1:n})\, dx_{1:n} = (1/N)\sum_{i=1}^{N} g(X_{1:n}^i)$. This standard approach may fail however, if the dimension n is large and π_n is highly complex, so that sampling is either impossible or too expensive, e.g., with costs proportional to n. Computation of the normalizing constant is often the culprit with respect to high cost.

Importance sampling methods try to address the foregoing problems using a simple but powerful idea: to sample from a complicated distribution, first sample from a tractable distribution and then "correct" the sampled value via a multiplicative *weight*. In particular, suppose that for some $n \geq 1$ we want to approximate a distribution π_n from which it is hard to sample and there exists a *proposal density* q_n (also called an *importance density*) such that (i) it is relatively easy to sample from q_n and (ii) $q_n(x_{1:n}) > 0$ whenever

$\pi_n(x_{1:n}) > 0$. Then, trivially,

$$\pi_n(x_{1:n}) = w_n(x_{1:n}) q_n(x_{1:n}) / Z_n \tag{1}$$

and

$$Z_n = \int w_n(x_{1:n}) q_n(x_{1:n})\, dx_{1:n}, \tag{2}$$

where w_n is the *unnormalized weight function*

$$w(x_{1:n}) = \gamma_n(x_{1:n}) / q_n(x_{1:n}).$$

Thus we can (easily) draw N independent samples $\{X^i_{1:n}\}_{1\le i\le N}$ from q_n and insert the resulting Monte Carlo approximation of q_n into (1) and (2) to obtain

$$\hat{\pi}_n(x_{1:n}) = \sum_{i=1}^{N} W^i_n \delta_{X^i_{1:n}}(x_{1:n})$$

and

$$\hat{Z}_n = \frac{1}{N} \sum_{i=1}^{N} w_n(X^i_{1:n}),$$

where the *normalized weights* are given by

$$W^i_n = \frac{w_n(X^i_{1:n})}{\sum_{j=1}^{N} w_n(X^j_{1:n})}.$$

The samples are often called *particles*. Note that this method requires a priori knowledge only of γ_n and not π_n, so there is no need to know the value of the (hard-to-compute) constant Z_n in advance.

A sequential version of the above procedure, called *sequential importance sampling* (SIS) can be applied when the goal is to approximate a sequence $\{\pi_n\}_{n\ge1}$ of probability measures of increasing dimension. SIS is recursive, so that only an $O(1)$ cost is incurred at each time point. The idea is to use an importance density having a Markov structure, i.e.,

$$q_n(x_{1:n}) = q_1(x_1) \prod_{k=2}^{n} q_k(x_{1:k} \mid x_{1:k-1}) \quad \text{for } n > 1,$$

which can be evaluated recursively. Using the Markov structure, some algebra shows that the unnormalized weights can also be computed recursively: $w_n(x_{1:n}) = w_{n-1}(x_{1:n-1})\alpha(x_{1:n})$, where

$$\alpha_n(x_{1:n}) = \frac{\gamma_n(x_{1:n})}{\gamma_{n-1}(x_{1:n-1}) q_n(x_n \mid x_{1:n-1})}.$$

The method as described so far has a severe drawback. As n increases the IS estimate involves the product of more and more random weights, which can cause the variance of the estimate to grow exponentially or can cause $\hat{\pi}_n$ to "collapse", in that one weight will tend to 1 while the rest tend to 0. A solution to this problem is to obtain a new sample of size N at the end of each iteration by resampling the foregoing set of N particles according to their normalized weights $W^1_n, \ldots, W^N_n$. Each element in the new set of particles is independently sampled and is assigned a weight of $1/N$, thus preventing collapse or exponential growth. Note that the new set of particles is a sample from $\hat{\pi}_n$, and hence approximately a sample from π_n. The resulting method is called sequential importance sampling with resampling (SIR).

We now return to the particle filtering method, which starts with a *hidden Markov model* (also called a *state space model*), comprising (i) a discrete-time Markov chain $\{X_n\}_{n\ge1}$ specified by an initial distribution $p_1(x_1)$ and transition probabilities $p_n(x_n|x_{n-1})$ for $n \ge 2$, and (ii) an *observation process* $\{Y_n\}_{n\ge1}$ with associated probabilities $p_n(y_n|x_n)$ for $n \ge 1$. The goal is to infer at each time n the conditional probability density $p_n(x_n \mid y_{1:n})$ of the true state, given the observations. The particle filtering algorithm is obtained by specializing the SIR algorithm. Specifically, take $\gamma_n(x_{1:n}) = p_n(x_{1:n}, y_{1:n})$, so that $\pi_n(x_{1:n}) = p_n(x_{1:n} \mid y_{1:n})$. Based on the foregoing discussion, we obtain Algorithm 2. It can be shown that the proposal density $q^*_n(x_n \mid x_{n-1}, y_n) \propto p_n(x_n \mid x_{n-1}) p_n(y_n \mid x_n)$ is "optimal" for this algorithm in that it minimizes the variance of the random weights.

Algorithm 2 Particle Filtering

1: Sample $\{X^i_1\}_{1\le i\le N}$ from $q_1(x_1 \mid y_1)$
2: Compute weights $w_1(X^i_1) = p_1(X^i_1) p_n(y_1 \mid X^i_1) / q_n(X^i_1 \mid y_1)$ for $1 \le i \le N$
3: Compute normalized weights $\{W^i_1\}_{1\le i\le N}$
4: Resample $\{(W^i_1, X^i_1)\}_{1\le i\le N}$ to obtain $\{(\frac{1}{N}, \bar{X}^i_1)\}_{1\le i\le N}$
5: **for** $n \ge 2$ **do**
6: Sample $\{X^i_n\}_{1\le i\le N}$ from $q_n(x_n \mid y_n, \bar{X}^i_{n-1})$
7: **for** $i = 1, 2, \ldots, N$ **do**
8: Compute weight $\alpha^i_n = p_n(y_n \mid X^i_n) p_n(X^i_n \mid \bar{X}^i_{n-1}) / q_n(X^i_n \mid y_n, \bar{X}^i_{n-1})$
9: **end for**
10: Compute normalized weights $W^i_n = \alpha^i_n / \sum_{j=1}^{N} \alpha^j_n$ for $1 \le i \le N$
11: Resample $\{(W^i_n, X^i_n)\}_{1\le i\le N}$ to obtain $\{(\frac{1}{N}, \bar{X}^i_n)\}_{1\le i\le N}$
12: **end for**

In [56], Xue et al. exploit the particle filtering algorithm to combine simulated data and sensor measurements. Their modified version of the DEVS-FIRE model simulates the stochastic progression of a wildfire over a gridded representation of terrain, where the current fire state records for each cell whether the cell is unburned, burning, or burned and, if burning, the intensity of the fire. The state of the fire after the nth simulation step and the corresponding sensor data correspond to x_n and y_n as above. The goal is therefore to compute the conditional density $p_n(x_n \mid y_n)$, i.e., to use the sensor data to "correct" the simulation or, looked at another way, to use the simulation to infer the state of the fire from the sensor data. The simulation steps correspond to increments of Δt simulated time units, where Δt is determined by the sensor measurement frequencies and the model's time-scale granularity. Based on scientific studies, the authors obtain a Gaussian model of sensor behavior, which leads to a closed-form expression for the observation function $p_n(y_n \mid x_n)$ as required in Steps 2 and 8. The original formulation in [56] uses the state transition probability $p_n(x_n \mid x_{n-1})$ as the proposal density q_n for $n > 1$ and uses $p_1(x_1)$ for q_1. With these choices, the formulas for the weights reduce to an evaluation of the observation function in Steps 2 and 8. Moreover, the task of sampling from $q_n(x_n \mid y_n, \bar{X}^i_{n-1})$ in Step 6 reduces to sampling from $p_n(x_n \mid \bar{X}^i_{n-1})$; this sampling is accomplished simply by setting the state of the simulation to $\bar{X}^i_{n-1}$ and then simulating for Δt time units. (The initial sampling in Step 1 is handled similarly.)

Note that, unlike the optimal proposal density q^*_n, the foregoing version of q_n ignores the sensor data y_n. Experiments with the model showed, perhaps not surprisingly, that accuracy degrades when the transition density $p_n(x_n \mid x_{n-1})$ is far from the optimal proposal density q^*_n mentioned above. The authors address this issue in [57], where they provide a proposal density that is sensitive to the sensor measurements. In brief, the process starts by first generating a fire state x from $p_n(x_n \mid x_{n-1})$ as described earlier. Then, based on sensor readings, another fire state x' is generated from x by (i) randomly igniting unburned cells in x that are deemed to have sufficiently high sensor temperatures and (ii) "turning off" the fire for x cells where sensor temperatures are deemed sufficiently cool. Then either x or x' is selected at random, according to a probability that is based on the relative "confidence" in the sensors and in the simulation model, and the selected state is

returned as the sample from q_n. To obtain analytical expressions for both $p_n(X_n^i \mid \bar{X}_{n-1}^i)$ and for $q_n(X_n^i \mid y_n, \bar{X}_{n-1}^i)$, as are needed to compute the weights in Step 8, $M > 1$ additional samples are drawn from these distributions using the methods discussed above and then the density functions are estimated using a standard *kernel density estimator* (KDE). For example, given samples $x_1, x_2, \ldots, x_M$ from the density $f_n(x) = p_n(x \mid \bar{X}_{n-1}^i)$, the density function is estimated as $\hat{f}_n(x) = (Mh)^{-1} \sum_{i=1}^{M} K\big((x - x_i)/h\big)$, where $h > 0$ is the KDE "bandwidth" and K is the KDE "kernel". The kernel is a nonnegative symmetric function such that $K(0) > 0$ and $K(x)$ is non-increasing in $|x|$, e.g., $K(x) = e^{-|x|}$; see [49] for a classical treatment and [15] for a discussion of more advanced kernel density methods. Finally, the method sets $p_n(X_n^i \mid \bar{X}_{n-1}^i) = \hat{f}_n(X_n^i)$ in Step 8, and the q_n term is handled analogously. Preliminary experiments indicate that the new proposal distribution can potentially lead to improvements in accuracy. As with model calibration, there are many opportunities for research in this area.

4. SIMULATION METAMODELING

We now focus our attention on the data generated from simulation models. Large, high-resolution models can easily generate terabytes of data during a run. Moreover, simulation models often have many input parameters, so there can be a combinatorially huge parameter space to explore, with large amounts of data being generated for each parameter-value combination simulated. For stochastic models, the amount of data generated is multiplied by the number of Monte Carlo replications. These challenges are exacerbated in composite modeling systems such as Splash. To fully exploit the potential of simulation models as tools for understanding complex systems, it is crucial that the generation of simulated data be carefully controlled and efficient.

Often, the first task in understanding a model is to identify the input parameters to which the model is most sensitive. Such sensitivity analysis can drastically reduce the size of the input parameter space by decreasing its dimensionality. Knowing which parameters are the most important can also guide the input-data collection process by focusing resources on data that yields sharper estimates of the important parameters. Often the parameters correspond to decision variables, and the goal is to identify optimal parameter settings that maximize or minimize some performance measure of interest. As discussed previously, the problem of model calibration falls into this category, where the performance measure to be minimized is the discrepancy between simulated and observed data. More generally, engineers and scientists have a set of questions that they want to answer using the model. One approach to controlling the amount of simulated data that is generated is to try and simulate just enough to capture the features pertinent to the questions of interest. Specifically, the use of statistical experimental-design methodology can reduce data-generation requirements by orders of magnitude. As discussed in what follows, the key concept underlying experimental design is simulation metamodeling.

4.1 Simple and Complex Metamodels

A simulation metamodel is a simplified functional representation of a simulation model, i.e., a response surface, that approximates the model response as a function of the input parameters. For a stochastic model, the response is often an expected value of some quantity of interest, such as profit. An appealing property of a metamodel is that is supports "simulation on demand": once a metamodel has been fit to the simulation data, then an approximation of the model output corresponding to given input values can be obtained almost instantly, allowing for exploration of a model in real time. For simplicity, we focus on real-valued responses throughout.

Metamodels vary in their complexity. The classic *polynomial model* relates the model response $Y(\mathbf{x})$ to the input parameters $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ via

$$\begin{aligned} Y(\mathbf{x}) = &\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \beta_{1,2} x_1 x_2 + \cdots \\ &+ \beta_{1,2,3} x_1 x_2 x_3 + \cdots + \beta_{1,2,\ldots,n} x_1 x_2 \cdots x_n + \varepsilon, \end{aligned} \tag{3}$$

where the β coefficients are real-valued constants and ε is a zero-mean random variable that encapsulates the stochastic variability. (Dropping ε in the above equation thus yields a model of the expected model response.) When only $\beta_0, \beta_1, \ldots, \beta_n$ are positive, we obtain a *linear model*, perhaps the simplest possible metamodel. (Sometimes, confusingly, the full polynomial model is referred to as a "linear model" because it is linear in the β coefficients.) The terms of the form $\beta_i x_i$ represent "main effects", whereas the remaining terms model second-order interaction effects, third-order effects, and so on.

At the other end of the complexity spectrum are *Gaussian process* metamodels. For deterministic models, perhaps the simplest form of such a model is

$$Y(\mathbf{x}) = \beta_0 + M(\mathbf{x}), \tag{4}$$

where the constant β_0 represents the mean response and $M(\mathbf{x})$ is a *stationary Gaussian process* (also called a *stationary Gaussian random field*), that is, a real-valued random process such that for any finite collection of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r$ the random vector

$$V = \big(M(\mathbf{x}_1), M(\mathbf{x}_2), \ldots, M(\mathbf{x}_r)\big)$$

has a multivariate normal distribution with, $E[V] = (0, 0, \ldots, 0)$. In applications, the covariance matrix is often defined as

$$\Sigma_M(\mathbf{x}_i, \mathbf{x}_j) = Cov[M(\mathbf{x}_i), M(\mathbf{x}_j)] = \tau^2 \prod_{k=1}^{n} \exp\big(-\theta_j (x_{i,k} - x_{j,k})^2\big) \tag{5}$$

for each i and j. When the simulation is run at a set of *design points* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r$, the stochasticity of M models the uncertainty associated with the output of the simulation when run at a point $\mathbf{x}_0$ that is not one of the design points. Given the observed model outputs at the design points, it can be shown [3] that the optimal estimator (in terms of minimizing mean square error) is

$$\hat{Y}(\mathbf{x}_0) = \beta_0 + \Sigma_M(\mathbf{x}_0, \cdot)^\top \Sigma_M^{-1} (\bar{Y} - \beta_0 \mathbf{1}_r), \tag{6}$$

where $\Sigma_M(\mathbf{x}_0, \cdot) = \big(\Sigma_M(\mathbf{x}_0, \mathbf{x}_1), \ldots, \Sigma_M(\mathbf{x}_0, \mathbf{x}_r)\big)$, the vector $\bar{Y}$ is given by $\big(Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_r)\big)$, the $r \times r$ matrix Σ_M is the covariance matrix of the design points, and $\mathbf{1}_r = (1, 1, \ldots, 1)$ is a vector of length r. The main observations are that $\hat{Y}(\mathbf{x}_i)$ coincides with the observed value $Y(\mathbf{x}_i)$ at each design point $\mathbf{x}_i$ and, for an arbitrary point $\mathbf{x}_0$, the centered estimator $\hat{Y}(\mathbf{x}_0) - \beta_0$ is a linear combination of $Y(\mathbf{x}_1) - \beta_0, Y(\mathbf{x}_2) - \beta_0, \ldots, Y(\mathbf{x}_r) - \beta_0$. In practice the various parameters that appear in (6)—such as β_0, Σ_M, and so on—are estimated from the data.

For stochastic simulations, the jth observation at design point $\mathbf{x}_i$ is modeled as in (4), except with an additional additive term $\varepsilon_j(\mathbf{x})$ that represents the random variability between simulation runs at a given design point. Here, for each i, the random variables $\varepsilon_1(\mathbf{x}_i), \varepsilon_2(\mathbf{x}_i), \ldots$ are i.i.d. normal with mean 0 and variance $V(\mathbf{x}_i)$, independent of M and of $\varepsilon_j(\mathbf{x}_h)$ for all j and $h \neq i$. The estimator $\hat{Y}(\mathbf{x}_0)$ is defined almost as in (6) above, but the ith element of $\bar{Y}$ is now the average result over all simulation runs at design point $\mathbf{x}_i$ and the term Σ_M^{-1} is replaced by $[\Sigma_M + \Sigma_\varepsilon]^{-1}$, where Σ_ε is the

Run	Parameters						
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	-1	-1	-1	1	1	1	-1
2	1	-1	-1	-1	-1	1	1
3	-1	1	-1	-1	1	-1	1
4	1	1	-1	1	-1	-1	-1
5	-1	-1	1	1	-1	-1	1
6	1	-1	1	-1	1	-1	-1
7	-1	1	1	-1	-1	1	-1
8	1	1	1	1	1	1	1

Figure 3: Resolution III design for seven parameters

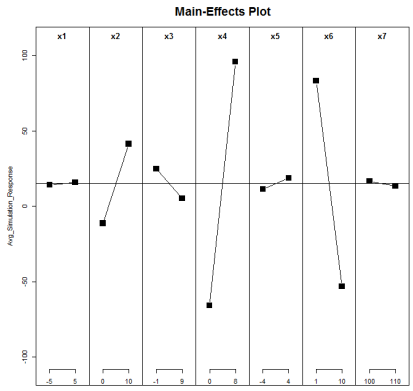


Figure 4: Main-effects plot for seven parameters

covariance matrix given by

$$\Sigma_\varepsilon(h,i) = Cov\Big[(1/n_h)\sum_{j=1}^{n_h}\varepsilon_j(\mathbf{x}_h),(1/n_i)\sum_{j=1}^{n_i}\varepsilon_j(\mathbf{x}_i)\Big]$$

and n_j denotes the number of Monte Carlo replications at $\mathbf{x}_j$.

Some good discussions of Gaussian process metamodels for deterministic and stochastic simulations can be found in [47] and [3], respectively. The basic ideas extend to more general settings where, for deterministic simulations, the metamodel has the form (4) but the random field M need not necessarily be Gaussian and the constant β_0 may be replaced by a more general regression model, e.g., as in (3). In this more general context, the metamodeling technique is often called *kriging*, after mining engineer D. G. Krige. The authors in [3] denote by *stochastic kriging* their extension of kriging obtained by adding a term ε_j, as discussed previously, to encompass stochastic models. See [44] for a recent example of stochastic kriging using a random field more complex than a basic Gaussian field.

4.2 Metamodeling and Experimental Design

The parameters of a metamodel encapsulate salient characteristics of simulation-model behavior. Selection of a particular metamodel leads to specific procedures for fitting the metamodel parameters. The power of experimental design lies in the observation that, if a relatively simple metamodel suffices to represent the simulated response, then the parameters of the metamodel—i.e., the key features of the simulation response—can often be estimated by exploring a very small but carefully selected subset of the parameter space, thereby reducing the amount of data that needs to be generated.

To illustrate, consider the polynomial model in (3) and suppose that $n = 7$. Of particular interest in this setting are the coefficients

Run	x_1	x_2
1	-4	-3
2	-3	4
3	-2	-1
4	-1	2
5	0	0
6	1	-2
7	2	1
8	3	-4
9	4	3

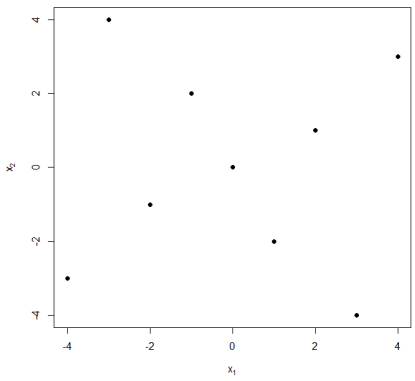


Figure 5: Latin hypercube design for two factors and nine runs

β_1 through β_7, which are called "main effects" or "sensitivities" and describe the change in simulation response corresponding to a given change in the parameter value while holding all other parameters constant. In classical experimental design, low and high values that represent ranges of feasibility or of problem-specific interest would be determined for each parameter, based on the experimenter's domain expertise. (Parameter values are usually called "factor levels" in experimental design terminology.) A naive "full factorial design" would then run simulations at all of the $2^7 = 128$ possible combinations of parameter values. Suppose, however, that prior knowledge leads one to believe that the higher-order terms in (3) can be ignored, so that a simple linear model adequately captures the shape of the response surface. One can then use a "resolution III fractional factorial design" as shown in Figure 3 to estimate the main effects using only eight simulation runs; in the figure, the symbols "-1" and "1" correspond to low and high values. For fractional factorial designs, the columns are orthogonal, which facilitates the ensuing statistical analysis. Note that, under the linearity assumption, the main effects capture virtually all model behavior of interest. Main effects are often displayed as in Figure 4. In this "main effects plot", each factor is characterized by two points, where the left (resp., right) point is the average simulation response over all runs where the parameter is set to its low (resp., high) value; these values are shown beneath the points. Main effects plots are typically accompanied with diagnostics that assess the statistical significance of the effect sizes, such as "half-normal plots", also called "Daniel plots" [14].

In a similar manner, if only third-order and higher effects can be ignored, one can estimate main effect using a resolution IV design that requires 16 runs. If the goal is to estimate both main and second-order effects, and if one can ignore third-order and higher effects, then a resolution V design requires only 32 runs. Thus experimental design methodology can be used to minimize the amount of data generated based on the both complexity of the response and the response characteristics of interest.

Rather than using only extreme values of the parameters as in fractional factorial designs, it is often desirable to use design points spread out evenly across the parameter space, especially when fitting complex nonlinear metamodels. A number of authors [3, 45] propose variants of *Latin hypercube* (LH) designs as providing a good compromise between covering the parameter space and minimizing the number of experiments. The basic procedure for a "randomized" LH with n parameters and $r \geq n$ design points—where r is typically of the form 2^k or $2^k + 1$ for some $k \geq 1$—is as follows. Determine r equally-spaced levels for each parameter and generate

an $n \times r$ design matrix where each column is a random permutation of $\{1, 2, \ldots, r\}$. Then the ith row gives the levels to use for the ith simulation run. Figure 5 shows a randomized LH design for $n = 2$ parameters and $r = 9$ levels (and design points), where the levels are designated as $-4, -3, -2, -1, 0, 1, 2, 3, 4$, along with a plot of the design points. The chief characteristic of an LH design is that each possible x_1 value appears once, as does each possible x_2 value. In general, randomized LH designs may not work well unless $r \gg n$. LH designs, however, are usually well behaved when the columns of the design matrix are orthogonal. (The LH design in Figure 5 is in fact orthogonal.) Because orthogonal designs can be rather hard to create, schemes for *nearly orthogonal LH* (NOLH) designs have been developed that provide good space-filling and orthogonality properties while being computationally efficient [12].

Because of its practical importance and theoretical elegance, the literature on experimental design is enormous. Treatments that are oriented toward stochastic simulation include the book of Kleijnen [34] and the tutorial paper of Sanchez and Wan [46]; the latter reference contains a nice table summarizing a wide variety of modern experimental designs. In [26], the authors describe the experiment management capabilities of the Splash composite-modeling platform, where metadata is used to provide an experimenter with a unified view of composite model parameters. Splash also provides a facility for specifying experimental designs as well as runtime support for setting parameter values by automatically synthesizing, via a templating mechanism, the input files that each component model expects.

4.3 Metamodeling and Factor Screening

Factor screening refers to the process of identifying the subset of parameters to which the simulation response is most sensitive. As mentioned previously, focusing on the key factors can greatly reduce the amount of generated data and experimentation effort. This problem is intimately related to metamodeling because, as we have seen, metamodel coefficients can quantify the sensitivity of the simulation output to changes in parameter values. Thus these metamodel coefficients can be used to classify model parameters as "important" or "unimportant".

For example, if a linear metamodel suffices, the observation noise can be modeled as Gaussian, and the main-effect coefficients are positive, then efficient *sequential bifurcation* methods can be used to identify important factors from among a potentially large set [50]. This type of procedure starts by dividing the set of parameters into two groups, and testing each group to decide if it contains at least one important parameter; such group testing is much faster than testing each individual parameter. If a group contains no important parameters, then it is discarded; otherwise, the group is again divided in two, and the testing procedure continues recursively.

For complex metamodels, the screening problem is much more difficult, and is a topic of ongoing research. Consider, for example, the Gaussian process metamodel in Section 4.1. It follows from (5) that a plausible measure for the importance of the jth factor is the coefficient θ_j: a very low value for θ_j implies a correlation function that approximately equals 1, so that there is no variability in model response as the value of the jth parameter changes. A number of studies have looked at the factor screening problem in this context; see, for example, [38].

5. CONCLUSION

To effectively support decisions in the enterprise, the information contained in big data must be combined with the information known to domain experts. Consequently, the fields of information management and of system simulation are intermingling more and more over time. Many of the questions concerning the interplay between models and data have not been formulated very precisely, and many of the techniques developed so far have been ad hoc. The PODS community is well poised to address the many issues around the increasingly important topic of model-data ecosystems.

Acknowledgments

The author wishes to thank Eric Bonabeau and Haidong Xue for providing materials and support and Ron Fagin for providing helpful feedback.

6. REFERENCES

[1] S. Alfarano, F. Wagner, and T. Lux. Estimation of agent-based models: the case of an asymmetric herding. *Comput. Econ.*, 26:19–49, 2005.

[2] T. T. Allen. *Introduction to Discrete Event Simulation and Agent-Based Modeling*. Springer, 2011.

[3] B. E. Ankenman, B. L. Nelson, and J. Staum. Stochastic kriging for simulation metamodeling. *Oper. Res.*, 58(2):371–382, 2010.

[4] Apache Hadoop. https://hadoop.apache.org.

[5] S. Arumugam, R. Jampani, L. Perez, F. Xu, C. Jermaine, and P. J. Haas. MCDB-R: Risk analysis in the database. In *VLDB*, pages 782–793, 2010.

[6] K. R. Bisset, J. Chen, S. Deodhar, X. Feng, Y. Ma, and M. V. Marathe. Indemics: An interactive high-performance computing framework for data-intensive epidemic modeling. *ACM Trans. Model. Comput. Simul.*, 24(1):4, 2014.

[7] E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proc. Nat. Acad. Sci.*, 99(3):7280–7287, 2002.

[8] E. Bonabeau. Big data and the bright future of simulation: The case of agent-based modeling. Keynote address, *Winter Simulation Conference*, December 2013.

[9] E. Bonabeau. Building accurate predictive models "without data". *Icosystem Blog*, accessed March 31. `http://www.icosystem.com/building-accurate-predictive-models-without-data`, 2014.

[10] K. Börner. Plug-and-play macroscopes. *Commun. ACM*, 54(3):60–69, 2011.

[11] Z. Cai, Z. Vagena, L. L. Perez, S. Arumugam, P. J. Haas, and C. M. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *SIGMOD*, pages 63–7–648, 2013.

[12] T. M. Cioppa and T. W. Lucas. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, 49(1):45–55, 2007.

[13] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. J. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S. Mckenna, B. D. Santer, and R. D. Smith. The community climate system model version 3 (CCSM3). *J. Climate*, 19:2122–2143, 2006.

[14] C. Daniel. Use of half-normal plots in interpreting factorial two-level experiments. *Technometrics*, 1(4):311–341, 1959.

[15] L. Devroye and G. Lugosi. *Combinatorial Methods in Density Estimation*. Springer, 2001.

[16] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later. In D. Crisan and B. Rozovskii, editors, *The Oxford Handbook of Nonlinear Filtering*. Oxford University Press, 2011.

[17] A. Fabretti. On the problem of calibrating an agent based model for financial markets. *J. Econ. Interact. Coord.*, 8:277–293, 2013.

[18] D. A. Ford, J. H. Kaufman, and I. Eiron. An extensible spatial and temporal epidemiological modelling system. *Int. J. Health Geographics*, 5(4), 2006.

[19] B. L. Fox and P. W. Glynn. Discrete-time conversion for simulating finite-horizon Markov processes. *SIAM J. Appl. Math*, 50(5):1457–1473, 1990.

[20] R. Franke and F. Westerhoff. Structural stochastic volatility in asset pricing dynamics: Estimation and model contest. *J. Econ. Dynam. Control*, 36:1193–1211, 2012.

[21] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, pages 69–77, 2011.

[22] P. W. Glynn and W. Whitt. The asymptotic efficiency of simulation estimators. *Oper. Res.*, 40(3):505–520, 1992.

[23] H. Godfray, J. Pretty, S. Thomas, E. Warham, and J. Beddington. Linking policy on climate and food. *Science*, 331(6020):1013–1014, 2011.

[24] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: From research prototype to industrial tool. In *SIGMOD Conference*, pages 805–810, 2005.

[25] P. J. Haas. Improving the efficiency of stochastic composite simulation models via result caching. 2014. Submitted for publication.

[26] P. J. Haas, N. C. Barberis, P. Phoungphol, I. Terrizzano, W.-C. Tan, P. G. Selinger, and P. P.Maglio. Splash: Simulation optimization in complex systems of systems. In *50th Allerton Conf.*, 2012.

[27] P. J. Haas, P. P. Maglio, P. G. Selinger, and W. C. Tan. Data is dead... without what-if models. *PVLDB*, 4(12):1486–1489, 2011.

[28] P. J. Haas and Y. Sismanis. On aligning massive time-series data in splash. In *VLDB Big Data Workshop*, 2012. Available at `researcher.watson.ibm.com/researcher/files/us-phaas/mta.pdf`.

[29] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Chapman and Hall, 1964.

[30] L. P. Hansen. Large sample properties of generalized method of moments estimators. *Econometrica*, 50(4):1029–1054, 1982.

[31] B. Howe and D. Maier. Algebraic manipulation of scientific datasets. *VLDB J.*, 14(4):397–416, 2005.

[32] T. T. Huang, A. Drewnowski, S. K. Kumanyika, and T. A. Glass. A systems-oriented multilevel framework for addressing obesity in the 21st century. *Preventing Chronic Disease*, 6(3), 2009.

[33] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. The Monte Carlo Database System: Stochastic analysis close to the data. *TODS*, 36(3):1–41, 2011.

[34] J. P. C. Kleijnen. *Design and Analysis of Simulation Experiments*. Springer, 2008.

[35] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, New Jersey, 1999.

[36] A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, sixth edition, 2014.

[37] E. L. Lehmann and G. Casella. *Theory of Point Estimation*. Springer, second edition, 1998.

[38] C. Linkletter, D. Bingham, N. Hengartner, and K. Q. Ye. Variable selection for Gaussian process models in computer experiments. *Technometrics*, 48(4):478–490, 2006.

[39] C. M. Macal and M. J. North. Introductory tutorial: Agent-based modeling and simulation. In *Proc. Winter Simul. Conf.*, pages 362–376, 2013.

[40] F. Makari, C. Teflioudi, R. Gemulla, P. J. Haas, and Y. Sismanis. Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion. *Knowl. Info. Sys.*, 2014. In press.

[41] D. McFadden. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica*, 57(5):995–1026, 1989.

[42] L. L. Perez, S. Arumugam, and C. M. Jermaine. Evaluation of probabilistic threshold queries in MCDB. In *SIGMOD*, pages 687–698, 2010.

[43] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22:400–407, 1951.

[44] P. Salemi, J. Staum, and B. L. Nelson. Generalized integrated brownian fields for simulation metamodeling. In *Proc. Winter Simul. Conf.*, pages 543–554, 2013.

[45] I. Salle and M. Yildizoglu. Efficient sampling and metamodeling for computational economic models. *Comput. Econ.*, 2013. DOI 10.1007/s10614-013-9406-7.

[46] S. M. Sanchez and H. Wan. Work smarter, not harder: a tutorial on designing and conducting simulation experiments. In *Proc. Winter Simul. Conf.*, page 170, 2012.

[47] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[48] T. C. Schelling. Dynamic models of segregation. *J. Math. Sociol.*, 1:143–186, 1971.

[49] D. W. Scott. *Multivariate Density Estimation: Theory Practice, and Visualization*. Wiley, 1992.

[50] H. Shen and H. Wan. A hybrid method for simulation factor screening. In *Proc. Winter Simul. Conf.*, pages 382–389, 2006.

[51] D. Shi and R. J. Brooks. The range of predictions for calibrated agent-based simulation models. In *Proc. Winter Simul. Conf.*, pages 1198–1206, 2007.

[52] V. Suryanarayanan and G. K. Theodoropoulos. Synchronised range queries in distributed simulations of multiagent systems. *ACM Trans. Model. Comput. Simul.*, 23(4):25, 2013.

[53] W. C. Tan, P. J. Haas, R. L. Mak, C. A. Kieliszewski, P. G. Selinger, P. P. Maglio, S. Glissmann, M. Cefkin, and Y. Li. Splash: a platform for analysis and simulation of health. In *ACM Intl. Health Informatics Symp. (IHI)*, pages 543–552, 2012.

[54] G. A. Wainer. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. CRC Press, 2009.

[55] G. Wang, M. Vaz Salles, B. Sowell, X. Wang, T. Cao, A. Demers, J. Gehrke, and W. White. Behavioral simulations in MapReduce. *Proc. VLDB*, 3(1):952–963, 2010.

[56] H. Xue, F. Gu, and X. Hu. Data assimilation using sequential Monte Carlo methods in wildfire spread simulation. *ACM Trans. Model. Comput. Simul.*, 22(4):23, 2012.

[57] H. Xue and X. Hu. An effective proposal distribution for sequential Monte Carlo methods-based wildfire data assimilation. In *Proc. Winter Simul. Conf.*, pages 1938–1949, 2013.