Design Space Exploration for Quantifying a System Model's Feasible Domain

Brad J. Larson
Ph.D. Candidate
Mem. ASME Student
e-mail: brad.larson@byu.edu

Christopher A. Mattson
Assistant Professor
Mem. ASME
e-mail: mattson@byu.edu

Department of Mechanical Engineering,
Brigham Young University,
Provo, UT 84602

A major challenge in multidisciplinary system design is predicting the effects of design decisions at the point these decisions are being made. Because decisions at the beginning of system design, when the least is known about the new system, have the greatest impact on its final behavior, designers are increasingly interested in using compositional system models (system models created from independent models of system components) to validate design decisions early in and throughout system design. Compositional system models, however, have several failure modes that often result in infeasible or failed model evaluation. In addition, these models change frequently as designs are refined, changing the model domain (set of valid inputs and states). To compute valid results, the system model inputs and states must remain within this domain throughout simulation. This paper develops an algorithm to efficiently quantify the system model domain. To do this, we (1) present a formulation for system model feasibility and identify types of system model failures, (2) develop a design space exploration algorithm that quantifies the system model domain, and (3) illustrate this algorithm using a solar-powered unmanned aerial vehicle model. This algorithm enables systematic improvements of compositional system model feasibility. [DOI: 10.1115/1.4005861]

1 Introduction

The motivation for quantifying a system model's feasible domain is rooted in a growing effort to reuse existing models of component behavior to predict system behavior. Initiatives by DARPA [1], NASA [2], Lockheed Martin [3], and Boeing [4] each involve designing and modeling new systems based on reusable components and component models (e.g., mechanics, propulsion, and electronics). These system models combine both the capabilities and the limits of the component models. This paper provides a method to quantify the resulting system model evaluation limits—the feasible domain. The design space exploration algorithm developed in this paper assists in system model verification by quantifying the system model's feasible domain and identifying the set of valid model simulation conditions where the system model can be used [5].

A *system model* is a model that predicts system behavior by combining various component models by model composition. *Model composition* is the process of combining component models by evaluating one model, passing its results to a second model, and then evaluating this second model [6]. *Simulation* is model evaluations used to determine the behavior of the system [7]. A *feasible domain* is the set of system model inputs and states from which the model produces valid results (e.g., the feasible domain of the linear spring model $f = k \cdot x$ is displacements x where the spring's response is linear) [8]. A simulation is feasible if it remains within the feasible domain. Evaluating a system model outside this feasible domain is an *evaluation failure*.

System models developed by model composition have several potential sources of failure, including each component model [9], each composition [10], data communication between models [11], and system model convergence [12]. For system models to be useful for verifying design decisions, they must produce results over the desired model domain and in the presence of design changes.

Model verification in practice today includes manual testing, automated test, vector execution, random test generation [8], and metamorphic testing where tests "mutate" to explore new areas of the system under test [10]. While these methods have proven to be effective for software and digital logic verification, they do not take advantage of the nature of system models. We propose that system model feasibility can be more effectively quantified based on known system model behavior and failure modes.

Sequential sampling techniques have proven to be efficient methods for design space exploration. The "efficient global optimization" algorithm built on Gaussian process regression has demonstrated both high accuracy and efficiency on a variety of data sets [13]. Kleijnen and Beers illustrate selectivity new samples based on the previous metamodel prediction [14]. Xiong et al. expand this work and combined Gaussian and Bayesian processes to introduce, among other objectives, uncertainty into the Gaussian process model [15]. Shan and Wang perform sequential sampling using radial basis functions to quantify the behavior of unknown functions for high dimensional problems [16].

To quantify system model feasibility, this paper also builds on several recent publications on design space exploration. Specifically, Huang and Chan employ Gaussian process regression to identify the feasible design space of various constrained functional designs [17]. Devanathan and Ramani, similarly, identify design space boundaries as a polytope (n-dimensional polygon) [18]. Finally, Malak and Paredis identify valid input domain boundaries of a fixed set of input data using support vector machines [19]. We extend these methods in two ways: (1) this paper addresses dynamic compositional system models in addition to functional models, and (2) it explores a binary space (system model domain) and identifies the boundaries of this system model domain.

In summary, this paper develops a design space exploration algorithm to quantify a system model's feasible domain, identifies portions of the system model domain where solutions exist, and enables failure boundaries to be classified based on failure mode. This algorithm can be used during system model validation to quantify and improve the feasibility of system models. Section 2 presents a formulation for system model feasibility and identifies types of system model evaluation failures. Section 3 develops a design space exploration algorithm to determine system model feasible domains and illustrates this using a solar-powered unmanned aerial vehicle (UAV) system model.

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received May 13, 2011; final manuscript received January 11, 2012; published online March 28, 2012. Assoc. Editor: Christiaan J. J. Paredis.

2 System Model Evaluation Failures

To be useful in design, a system model must compute valid results. Because the nature and behavior of compositional system models are not precisely defined, Secs. 2.1 and 2.2 briefly present a formal definition of system models, component models, and discipline-specific models that are able to represent most systems and components found in engineering design. Based on these definitions and equations, we identify various types of evaluation failures exhibited by system models in Sec. 2.3.

2.1 System Model Formulation. A system model

$$\boldsymbol{y} = S(\boldsymbol{u}) \quad \text{where} \quad \{\boldsymbol{u} \in \boldsymbol{U}\} \quad \text{and} \quad \{\boldsymbol{y} \in \boldsymbol{Y}\} \tag{1}$$

is a relation producing the output sequence $\boldsymbol{y}$ from the range $\boldsymbol{Y}$ based on the input sequence $\boldsymbol{u}$ from the domain $\boldsymbol{U}$. For example, the input sequence $\boldsymbol{u}$ for an UAV could be throttle input. The output sequence $\boldsymbol{y}$ could be the resulting propeller thrust.

A system model S (Eq. (1)) predicts the response to stimulus of a physical system by combining various component models by composition [20]. Model composition is a one-way data transfer from the output of one model to the input of another. Figure 1 illustrates a system model developed by composition of three component models, where component models are the boxes C_1, C_2, and C_3. Composition is data transfer between component models and is shown as arrows in Fig. 1. The solution to system models can be found by transferring data and evaluating component models as established by the system model graph. Loops in the system model, as illustrated in Fig. 1, may require iterative convergence [21].

2.2 Component Model Formulation. A component model

$$y_j = C_j(u_j) \quad \text{where} \quad \{u_j \in U_j,\ y_j \in Y_j\} \tag{2}$$

is a relation that describes the behavior of one portion of a system. It is able to be combined by composition with other component models into a system model. The index j in Eq. (2) indicates that it is one of a set of component models that comprises a system model.

Our focus is on component models that enable discipline-specific engineering models (e.g., CAD models, dynamics models, digital logic, computer algorithms, etc.) to be combined into system models. We represent these discipline-specific engineering models as

$$D_j = (U_j, X_j, Y_j, f_j, h_j, x_{j,0}) \tag{3}$$

where

U_j: set of inputs	$\{u_j \in U_j\}$
X_j: set of states	$\{x_j \in X_j\}$
Y_j: set of outputs	$\{y_j \in Y_j\}$
f_j: progression function	$f_j: U_j \times X_j \rightarrow X_j$
h_j: output function	$h_j: U_j \times X_j \rightarrow Y_j$
$x_{j,0}$: initial state	$\{x_j, 0 \in X_j\}$

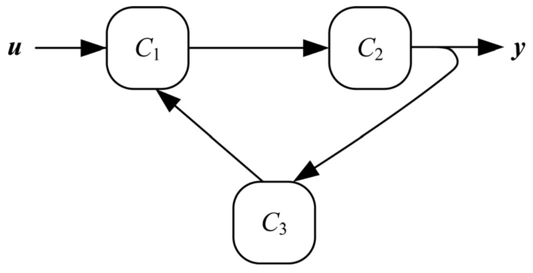

Fig. 1 Example of a compositional system model

which is a general dynamic system producing outputs y_j based on inputs u_j. The parentheses in Eq. (3) denote a tuple, which is an ordered collection of possibly heterogeneous elements. Equation (3) maps from the input set U_j, through the set of states X_j, to the model output set Y_j. These mappings are accomplished by both the progression function f_j and the output function h_j. The progression function f_j maps from one location to another within the state space X_j, based on the current input u_j and state x_j. This is written as f_j: $U_j \times X_j \rightarrow X_j$ (Eq. (3)). The notation $U_j \times X_j$ is the Cartesian product of the sets U_j and X_j, which means the input to f_j can be any combination of $u_j \in U_j$ and $x_j \in X_j$. The arrow ($\rightarrow$) denotes the function's mapping of these inputs to its states $x_j \in X_j$. The output function h_j maps from the current input set U_j and set of states X_j to the output set Y_j (h_j: $U_j \times X_j \rightarrow Y_j$ in Eq. (3)). $x_{j,0}$ is the initial state of x_j.

This dynamic system formulation of component models (Eq. (2)), which we restrict to representing the discipline-specific models (Eq. (3)), is able to represent most types of models used in engineering system design such as algebraic relationships, data look-up, stochastic processes, frequency domain models, differential-algebraic equations, infinite-dimensional dynamic systems, discrete-event systems, and computer programs [21,22]. For example, systems of algebraic equations or data look-up can be represented as the functional mapping from an input space U_j to an output space Y_j with an empty state space using only the output function: h_j: $U_j \times X_j \rightarrow Y_j$. A system of linear time-invariant differential equations $\dot{x}(t) = Ax(t) + Bu(t)$ and $y(t) = Cx(t) + Du(t)$ can be formulated using Eq. (3) with f_j as the integration of $Ax(t) + Bu(t)$ and the function h_j as $Cx(t) + Du(t)$.

2.3 Evaluation of Failure Sources. Based on the model formulations above, Table 1 first identifies failure modes for component models (which include discipline-specific models). System model failures combine all component model failures as described by the component model composition and introduce several additional failure modes. Because the reuse of existing discipline-specific models includes both the component model definition and solution, we have not distinguished between model definition failures and model evaluation failures (e.g., errors introduced by the solver). The failure type column describes how these failures behave in relation to model evaluation. Specifically, *dynamic failures* are deterministic failures that depend both on model inputs and model states, *static failures*(a subset of dynamic) are deterministic failures that depend only on model inputs, *constant failures* are deterministic failures that do not depend on model inputs or states, *stationary failures* are probabilistic failures that are described by a constant statistic (e.g., data communication between models is 99.999% reliable), and *dynamic statistical failures* are probabilistic failures that are described by a changing statistic (e.g., the likelihood of failure is proportional to the model state x).

Table 1 Component and system model failures

Component failure	Description	Failure type
Domain	$u_j \notin U_j$ (Eq. (2))	Static
State	$x_j \notin X_j$ (Eq. (3))	Dynamic
Computation	f_j or h_j failure (Eq. (3))	Dynamic
Resource	Evaluation resources unavailable	Dynamic statistic
Development	Improper definition	Constant
System failure	Description	Failure type
Component	Any component failure Ref. [6]	Component failures
Convergence	Fails to converge Ref. [23]	Dynamic
Communication	Data transfer failure Ref. [24]	Stationary
Formulation	Unable to define $\boldsymbol{y} = S(\boldsymbol{u})$	Constant
Composition	Incorrect composition Ref. [25]	Constant
Range	$y \notin \boldsymbol{Y}$ (Eq. (1))	Dynamic

Because this failure classification focuses on the behavior of a failure during simulation, it differs from other model failure characterizations describing failures in terms of model structure. For example, a structural singularity describes a system model structural error resulting in a behavioral formulation failure. An algebraic loop in the model structure could result in convergence failure during evaluation. Stiffness or instability in component or system models could result in state or range failures during model evaluation. This focus on the behavior, rather than the structure of failures, enables us to develop a behavioral model of the feasible system model domain.

In this paper, we focus specifically on static and dynamic failures. The constant and stationary failures can be effectively addressed by techniques available from reliability research [8,26]. Resource failures (dynamic statistic) can be addressed directly by adding the necessary resources.

Quantifying system model feasibility due to static and dynamic failures is the process of finding the intersection of the valid component model inputs U_j, component model states X_j, valid computation, model convergence, and valid system model range $\boldsymbol{Y}$. Model composition, however, does not typically produce a geometric combination of feasible spaces. Rather, it is a combination defined by the structure and dynamics of the system model. While this hinders an analytical computation of the system model feasible space, we can discover this feasible space computationally.

Some important considerations for investigating dynamic failures, which include static failures, are: (1) a failure at any point results in a system model failure, (2) failures are interdependent, and (3) a failure may be induced by a preceding action from another part of the system model.

We next develop a domain exploration algorithm to determine system model feasibility.

3 Feasible Domain Exploration

To validate design decisions using system models, feasible solutions $\boldsymbol{y}$ must exist for the input sequences $\boldsymbol{u}$ that the physical system will encounter. In this section, we develop a design space exploration algorithm to identify the domain where valid system model solutions exist. This enables us to quantify and improve system model feasibility due to dynamic system model failures.

System model feasibility exploration is a tool to improve system model verification before the model is used in system design. The computational expense of formally quantifying system model feasibility should be offset by more effective utilization of the system model once the feasible domain is known during the following, more computationally expensive design, analysis and optimization processes.

To quantify system model feasibility, Sec. 3.1 presents a formulation for the feasibility of system models, then Sec. 3.3 develops a design space exploration algorithm. This method is illustrated using a solar-powered UAV propulsion system described in Sec. 3.2. Section 3.4 classifies the valid design space boundaries based on the failure mode. Section 3.6 identifies methods to improve system model feasibility.

3.1 Defining a Searchable Space. Because system models are general dynamic systems, feasible solutions depend on both the input magnitude and sequence. For example, a sequence that successfully lands a UAV would result in mostly disastrous landings if the sequence was arbitrarily rearranged. Since the system model input domain $\boldsymbol{U}$ includes all possible values of each element of the sequence, the full system model domain cannot be effectively explored. Most systems, however, have a much smaller set of useful input sequences. This section presents a method of establishing a searchable input domain for feasibility exploration by composition of the system model with two functions: an input function and a feasibility function.

The input function is

$$\boldsymbol{u}_\nu = I(\nu) \quad \{\nu \in N \subset \Re^n\}, \{\boldsymbol{u}_\nu \in \boldsymbol{U}_\nu \subset \boldsymbol{U}\} \tag{4}$$

where ν is an input configuration parameter array and $\boldsymbol{u}_\nu$ is the system model input sequence. The input function configuration parameter ν provides a continuous space suitable for evaluating model feasibility even for discontinuous and event-based system models. For example, a UAV throttle command could be any arbitrary sequence of commands, most being inappropriate for UAV flight. The input function $I(\nu)$ in this example confines system model input $\boldsymbol{u}$ to a set of sequences common to UAV flight such as take-off, climb, cruise, descent, and landing. The input parameter array ν configures these sequences.

The feasibility function interprets the system model evaluation and output sequence as being valid (1) or invalid (0). The feasibility function

$$F(S(\boldsymbol{u})) = \{1, 0\} \tag{5}$$

receives as input the system model and the input sequence it is to evaluate. It then evaluates the system model and determines if the model evaluation is valid. It evaluates the system model and produces a "1" if the system model S was able to compute valid results and "0" if it was not. This requires both range checking and process protection in system and component model execution, which was addressed in our previous work [27].

The composition of the feasibility function (Eq. (5)), system model (Eq. (1)), and input function (Eq. (4)) produces $F(S(I(\nu))) = \{1,0\}$, which enables functional system model evaluation over the continuous space N to be evaluated as either a valid or invalid simulation. We will refer to the set inputs ν where the feasibility function returns 1 as the feasible solution set F_ν. With this, we define the system model feasibility ratio as

$$R = \frac{\int \ldots \int F(S(I(\nu)))\, d\nu}{\int \ldots \int d\nu} \tag{6}$$

which is the hyper-volume ratio of the feasible design space to the total design space.

The challenge our design space exploration algorithm must solve is to efficiently quantify the size of the feasible solution set F_ν, when F_ν can only be evaluated one configuration ν at a time and when each model evaluation can be computationally expensive. Before addressing this, Sec. 3.2 introduces the UAV system model that is used to illustrate this algorithm.

3.2 UAV System Model. This section introduces a solar-powered UAV propulsion system model with which we will illustrate feasibility exploration. This model can exhibit all of the dynamic and static failures in Table 1. The UAV system model is segmented into four component models shown in Fig. 2: solar panel, battery, motor, and propeller. The system model has two inputs (the throttle command and initial battery charge) and produces a single output (thrust).

The first step to prepare this UAV system model for feasibility exploration is to ensure component models are only evaluated within their valid domain $(u_j \in U_j)$ and state $(x_j \in X_j)$. If system model execution exceeds these bounds, execution is terminated. This is usually a simple comparison of the input and state to their valid ranges. We next create appropriate input and feasibility functions.

For valid UAV model execution, the following conditions must be satisfied:

(1) the solar panel model must be from 6.6 V to 7.4 V (domain failure)
(2) the battery model must be from −20 A to 1 A (domain failure)

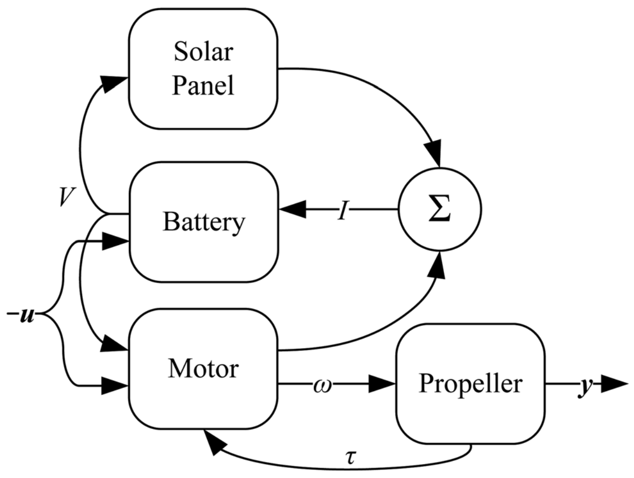

Fig. 2 Solar-powered UAV propulsion system model

(3) the battery model must be from 1.0% to 100.0% of its total charge (state failure)
(4) the motor torque must be ≤100 Nm (domain failure)
(5) the motor current must be ≤20 A (state failure)
(6) the propeller model speed must be from 0 to 95 Hz (domain failure)
(7) component models must compute solutions (component failure)
(8) the system model must converge to a solution (convergence failure)
(9) propeller thrust must be sufficient for take-off (range failure).

These ranges of valid model execution were readily available during component model development and should be captured for each component model as part of system model development.

Accounting for these failures, the feasibility function

$$F_{\mathrm{UAV}}(S(\boldsymbol{u})) = \begin{cases} 1 & \text{if } \int_0^{t_{\max}} dy > T \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

produces a 1 if the various failure modes are avoided and if the integral of the thrust is greater than the threshold T within the take-off duration $t_{\max.}$ The input function

$$I(\nu) = \begin{pmatrix} \nu_1 \\ C(\nu_2) \end{pmatrix} \tag{8}$$

defines the system model inputs: the initial battery charge (ν_1) and the throttle command waveform $C(\nu_2)$. The throttle command $C(\nu_2)$ is a piecewise-linear function that linearly interpolates between a fixed low and an adjustable high throttle speed.

The system model, feasibility function, and input function shown here for the solar-powered UAV propulsion system can be developed for most engineering systems. Although developing system models is typically challenging, creating input and feasibility functions should not be difficult. These two functions capture design information that is a standard part of system design specifications: system inputs and performance ranges. These functions produce a searchable input domain and a binary output range for evaluating system model feasibility.

3.3 Model Feasibility Design Space Exploration. Using the compositional system model failure modes identified in Table 1, this section develops a design space exploration algorithm to efficiently find the set of feasible solutions to the system model and to quantify system model feasibility (Eq. (6)). The set of model solutions, referred to as the feasible solution set F_ν, and its boundaries describe where model solutions exist and how to improve system model feasibility.

Several properties of system model feasibility aid the development of a design space exploration algorithm. The first property is that feasible solutions are part of closed sets but may not be connected sets. This means that the boundary between feasible and infeasible solutions is within the feasible solution set though, the set may have holes or may be multiple isolated regions. Second, the system model solution is either feasible 1 or infeasible 0 with no gradient. As a result, any slope between feasible and infeasible regions is a prediction error. Next, because we know that (1) boundaries are within our feasible solution set, (2) all values within this set are 1, and (3) all values outside are 0, the feasible solution set can be described by its boundaries. As such, the design space exploration algorithm searches for the feasible solution set boundaries.

To quantify system model feasibility based on these properties, we propose a design space exploration algorithm that builds upon other sequential sampling methods [17]. This method, illustrated in Fig. 3, performs the following steps:

(1) sample the input space $\boldsymbol{U}_\nu$ at random until both feasible and infeasible samples exist,
(2) estimate the feasible design space by linearly interpolating available samples,
(3) identify the most gradual transitions between the feasible and infeasible regions to refine,
(4) identify the least explored areas of design space $\boldsymbol{U}_\nu$ to refine,
(5) prioritize the next samples by combining transition and unexplored area metrics,
(6) add new samples at the highest scoring locations,
(7) exit based on number of samples or the stability of the model feasibility metric,
(8) repeat this sequence from step 2 until the feasibility estimate has stabilized or the maximum number of samples is reached.

The implementation in this paper addresses deterministic failures of system models but could be extended to address statistical failures.

The first step is to sample the input space at random until both valid and invalid samples exist. Latin hypercube sampling is used, because it ensures that samples are spread over the input space by selecting new sample locations at random from distinct segments of the input space [28]. This is typically just a few samples to seed the Gaussian process regression design space estimation.

With a sample population containing both valid and invalid results, the second step is to estimate the valid design space. Gaussian process regression (kriging) provides an accepted and flexible method of estimating the design space. It only requires an arbitrary set of sample locations ν and system responses $f(\nu_i)$ as input with which it predicts both the system response $\tilde{f}(\nu)$ and estimates the prediction error $\varphi(\nu)$ [29]. The predicted system response

$$\tilde{f}(\nu) = \sum_{i=1}^{n} \lambda_i(\nu) f(\nu_i) + Z(\nu) \tag{9}$$

estimates system model validity $\tilde{f}(\nu)$ for unexplored model inputs ν [29]. The weighting functions $\lambda_i(\nu)$ and zero-mean stochastic process $Z(\nu)$ are determined as part of Gaussian process regression. Prediction error

$$\varphi(\nu) = E[(\tilde{f}(\nu) - f(\nu))^2] \tag{10}$$

is defined as the expected value (E) of squared error between the predicted value $\tilde{f}$ and actual value f at location ν. It is also estimated as part of Gaussian process regression using the set of system responses $f(\nu_i)$ as described and implemented by Lophaven et al. [30].

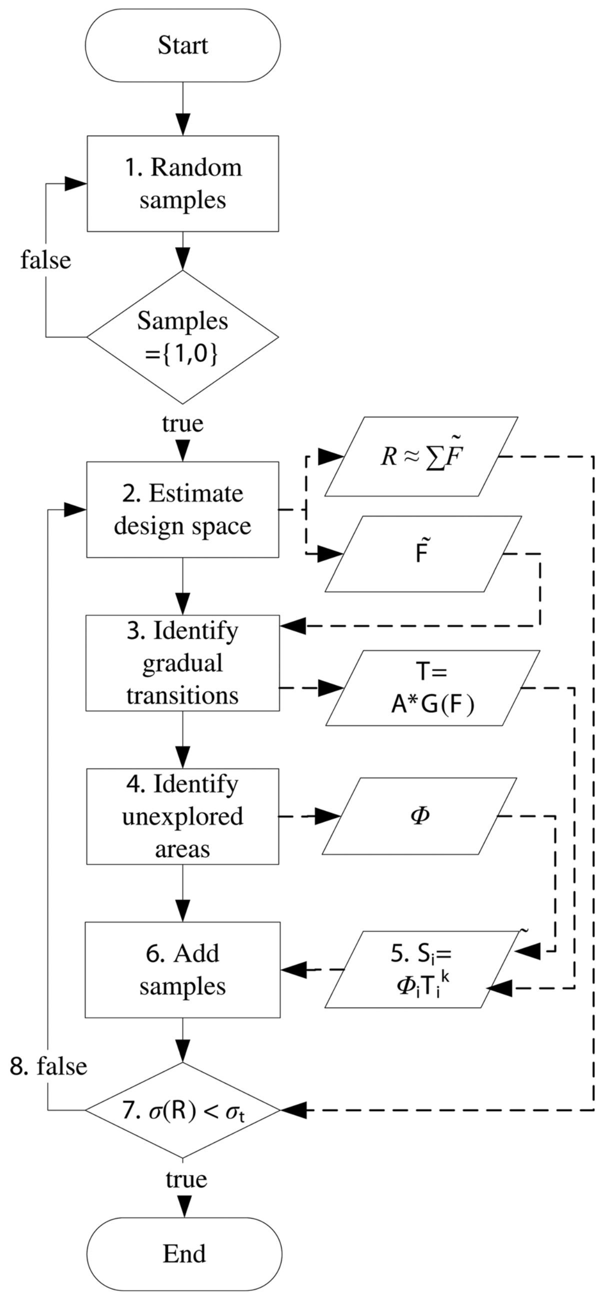

Fig. 3 Design space exploration algorithm overview

Using the predicted response and error, we next sample $\tilde{f}(\nu)$ and $\varphi(\nu)$ at regular intervals from the maximum to the minimum value of each input dimension of ν to produce the n-dimensional prediction matrix $\tilde{F}$

$$\tilde{F} = \tilde{f}(\nu_n) \tag{11}$$

and the error matrix

$$\Phi = \varphi(\nu_n) \tag{12}$$

Filtering the prediction matrix $\tilde{F}$ produces the binary feasibility space estimate

$$\tilde{F}_b = \begin{cases} 1 & \text{if } \tilde{F} >= 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

with 0.5 as the threshold between valid and invalid samples. The number of dimensions of each of these matrices is the number of configuration parameters in ν. The number of elements in each dimension establishes the prediction and error resolution.

The prediction matrix $\tilde{F}$ is used in step 3 to identify transitions between valid and invalid regions. The estimated error Φ is used by step 4 to identify unexplored areas in the design space.

The average value of $\tilde{F}_b$ estimates system model feasibility ratio

$$R \approx \frac{\sum \tilde{F}_b}{\prod j} \tag{14}$$

defined in Eq. (6). The number of elements in each dimension of $\tilde{F}$ is j.

The third step is to identify the most gradual transitions between the valid and invalid regions. Although the true boundary of the valid region changes immediately from 1 to 0, the feasibility space prediction $\tilde{F}$ interpolates between sampled locations. Consequently, a slowly changing transition in $\tilde{F}$ means the valid region boundary is not well defined. To identify gradual transitions, we will first identify the transition regions and then attenuate the quickly changing transitions.

Gradient-based edge detection, which is used in image processing, would be ineffective at finding gradual transitions in $\tilde{F}$ because both valid and invalid regions contain gradually changing values. Because we are looking for transitions in a binary space, we can simply look for transitions through the 0.5 threshold. To do this, we multiply each element of the predicted response by the Gaussian weighting function

$$G(V, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{15}$$

which amplifies the feasible solution set boundary.

Although the Gaussian weighting function identifies transitions, it weights them equally. To identify slowly changing boundaries, an averaging convolution kernel (shown as a two-dimension $m \times n$ convolution kernel in Eq. (16)) is convolved (written as *) with the results of the Gaussian weighting function [31]. This results in a spatial average over adjacent model predictions. For slowly changing transitions, adjacent values of $\tilde{F}$ will receive a similar value from the Gaussian weighting function. The averaging convolution will, consequently, have little effect on slow transitions but will attenuate sharp transitions more strongly.

$$A = \frac{1}{mn} \cdot \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix} \tag{16}$$

Combining these operations produces the n-dimensional model transition matrix

$$T = A * G(\tilde{F}, \mu, \sigma) \tag{17}$$

whose maximum values are the slowest model transitions.

The fourth step is to identify the least explored areas of the input space $\boldsymbol{U}_\nu$. The feasibility space error (Eq. (12)) provides an estimate of error that increases as the distance from samples increases. This can be used directly to identify the least explored areas of the design space.

The fifth step determines the next sample(s) by combining transition and unexplored area matrices. The resample priority matrix

$$P_i = \Phi_i T_i^k \quad \forall\, i \in \nu_m \tag{18}$$

is created by the element-by-element multiplication of the feasibility error matrix Φ and the transition matrix T raised to the power of k. Because the error is 0 at existing sample locations, multiplying by Φ ensures that new samples are not from previously sampled locations. To prioritize refining boundaries over exploring unexplored regions, k can be set to a value greater than 1. The sixth step adds new samples at the location of the maximum values of P.

The seventh step tests the exit criteria. Feasibility exploration can either exit based on the number of samples or the standard deviation of the feasibility prediction over some number of previous samples. If the standard deviation is less than a threshold

$$\sigma(R_{n-j}, \ldots, R_n) < \sigma_t \tag{19}$$

feasibility exploration can terminate. A larger threshold results in a faster search but less accurate boundaries. A σ_t of 0.01 would exit after the 37th sample in Fig. 5.

The eighth step continues feasibility exploration if the exit criteria are not satisfied.

Figure 4 shows the result of the first 40 samples of the feasibility exploration using the UAV model. The configuration parameters ν are the throttle command (vertical axis) and the initial battery charge (horizontal axis). Figure 4(*a*) shows the prediction matrix $\tilde{F}$. The light region in the center of this plot is the feasible solution set F_ν. The circles are the samples taken to identify the feasible solution set. These circles are shaded according to the failure type. Dark circles (all within the feasible region) indicate a successful simulation.

The UAV model's feasible region, shown in Fig. 4(*a*), is due to interactions within the model and cannot be fully captured by a fixed range of throttle command and initial charge. Although the feasibility search algorithm is able to find multiple disconnected regions, the UAV model valid region is a simply connected set, which is typical of the feasible regions we evaluated.

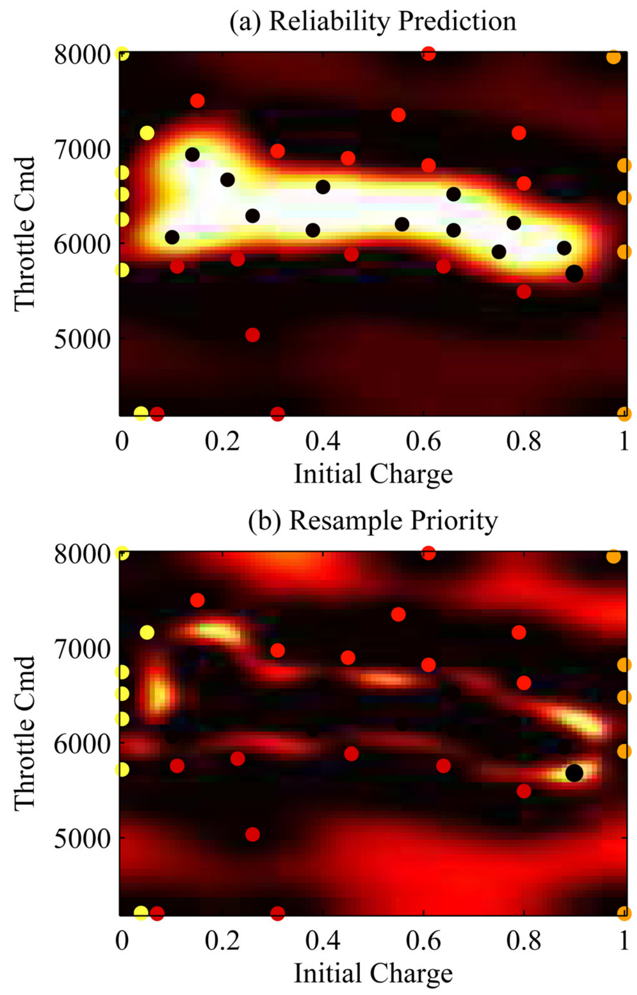

Fig. 4 Feasibility exploration of a UAV system model

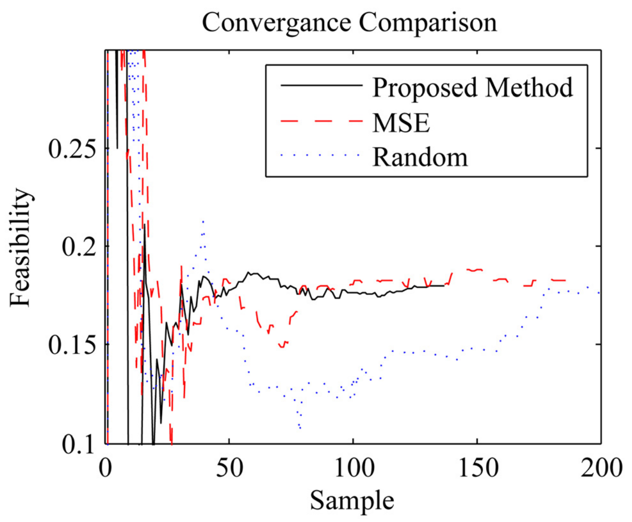

Fig. 5 Comparison of random, mean-squared error, and proposed feasibility design space exploration

Figure 4(*b*) shows the resample priority matrix P (Eq. (18)). The larger black circle ("throttle Cmd" $\approx$ 5700 and "charge" $\approx$ 0.9) has the greatest resample priority and is therefore the location of the next sample. We set $k = 2$ in Eq. (16) in order to preferentially refine known boundaries over exploring new areas for this example.

Feasibility exploration estimates system model feasibility using Eq. (14). The solid trace in Fig. 5 shows the change in the feasibility estimate versus the number of samples. Both the feasibility estimate and the feasible solution set boundary stabilize in about 40 samples.

3.4 Method Evaluation.

The input function (Eq. (4)) and feasibility function (Eq. (5)) enable various design space exploration algorithms to evaluate the feasibility of dynamic compositional system models. In this section, we compare the evaluation of three different design space exploration algorithms by comparing their computed feasibility (Eq. (14)). Our objective is to show that this proposed method converges to the same feasibility value at an acceptable rate.

To do this, we compare the proposed method to Latin hypercube random sampling and mean-squared error sampling. Both of these sampling methods perform unbiased global searches. The feasible space of each of these sampling strategies is interpolated using Gaussian process regression. Latin hypercube random sampling produces random samples that are guaranteed to cover all portions of a space. Therefore, it is preferred over pure random sampling for design exploration. Mean-squared error sampling chooses the next sample at the location farthest from existing samples.

Figure 5 plots the convergence of the feasibility metric of these three sampling methods over the number of samples using the solar-powered UAV system model. Each method converges to a similar feasibility value. The method developed in this paper converges to within 1% of the final solution within 40 samples. Mean-squared error achieves a similar accuracy in 80 samples. Latin hypercube sampling converges to this same feasibility value after 178 samples. The faster rate of convergence of this feasibility search illustrates the value of searching based on our a priori model of the feasible system model domain.

3.5 Feasible Solution Set Boundary Classification.

In addition to quantifying feasibility and identifying the feasible solution set, the feasible solution set boundaries can be classified based on the nearest failure. For example, in Fig. 4(*a*), the bottom boundary is due to the thrust range failure, the top boundary is the propeller valid speed domain failure, the left boundary is the battery current domain failure, and the right boundary is the solar panel voltage domain failure defined in Sec. 3.2. While this boundary classification can be accomplished visually for feasibility spaces with three or fewer dimensions, our future work addresses automatic boundary identification.

Categorizing the feasible solution set boundary in this way enables failures that define the system model feasible solution set to be identified. This also allows system model feasibility improvements to be prioritized. For example, if the valid range of propeller speeds were increased, the top boundary of the feasible solution set in Fig. 4(*a*) would rise, the size of the feasible solution set would increase, and system model feasibility would increase.

3.6 Increasing Feasibility Using System Model Feasibility Exploration. Feasibility exploration facilitates integrating compositional system models into system design. To use feasibility exploration in system design:

(1) system models are developed to only produce results within the valid model ranges U_j, X_j, etc.,
(2) input and feasibility functions are defined to produce a continuous, searchable input space and binary response for the system model: $F(S(I(\nu)))$,
(3) algorithm parameters k (exploration priority), σ_t (termination threshold), and maximum samples are set, and
(4) feasibility exploration is performed and recorded periodically during system design.

Using feasibility exploration, system model developers can qualify system model execution, identify valid input ranges, determine primary failure modes, identify model changes that reduce feasibility, and progressively improve system model feasibility.

The feasibility estimate alone can serve as a threshold to qualify and compare system models. If feasibility is too small, it is unlikely that a system model will effectively predict system behavior, especially in the presence of design changes. In addition, comparing the feasibility of different versions of a system model can identify the impact of specific model changes on system model feasibility.

The feasible solution set provides a more complex view into system model feasibility, as is seen in Fig. 4(*a*). Specific simulation conditions can be compared to the feasible solution set to determine whether the needed simulation conditions will evaluate correctly. Identifying the failure type along the feasible solution set boundaries indicates the source of system model failures. The larger the boundary of a specific failure mode, the more significant the failure mode. These measurements enable model improvements to be prioritized in order to increase the feasible solution set and determine needed simulation conditions. For example, a faster UAV take-off requires a greater throttle command and larger initial charge. This is limited by system model failures due to the propeller model's valid speed range. By increasing this range, system model feasibility would grow and new simulation conditions would become feasible.

4 Summary and Future Work

For system models to be useful in system design, they must produce feasible results over the desired model domain and throughout the entire design process. This is especially important for compositional system models due to their additional sources of failure. Of the many types of system model failures, dynamic and static failures, which are not well addressed by reliability research, have been the focus of this paper.

In this paper, we presented a formulation for system model feasibility and developed a design space exploration algorithm to identify the feasible system model domain. This enables developers to (1) determine whether solutions to specific simulation conditions exist, (2) identify if changes in a system model affect feasibility, (3) identify significant sources of system model failures, and (4) select system model improvements that will lead to feasibility improvements.

Our future work into system model feasibility includes extending feasibility exploration to statistical failures, quantifying the relationship between computational cost and the feasibility space dimension, classifying failure importance based on boundary size, and quantifying feasibility sensitivity to design changes.

Our initial studies into evaluating and improving the high-dimensional performance of the proposed design space exploration found that the growth in number of samples appears to be less than exponential with respect to the number of dimensions. Other computational enhancements, however, would benefit high dimensional design space exploration. For example, the literature recommends radial basis functions over Gaussian process regression for high dimensions [16]. In addition, Eq. (18) should be represented by a sparse, rather than a full matrix, to reduce the growth in data size. Although good high dimensional performance is important, the number of dimensions of feasibility design space exploration is the number of the input function parameters (ν), not size of the system model domain or number of model failures. Therefore, the search dimensions are likely to grow much more slowly than the complexity of system models.

Nomenclature

$y = S(\boldsymbol{u})$ =	system model
C_j =	jth component model within a system model
D_j =	jth discipline-specific model
$F(S(\boldsymbol{u}))$ =	feasibility function
ν =	input configuration parameters
$I(\nu)$ =	input function
F_ν =	feasible solution set
R =	system model feasibility ratio
$\tilde{F}(\nu)$ =	feasibility prediction matrix
$\Phi(\nu)$ =	feasibility error matrix
P =	resample priority matrix

References

[1] DARPA, T. T. O., 2009, "META," DARPA Tactical Technology Office, Technical Report No. DARPA-BAA-10-21.
[2] Gray, J., Moore, K. T., and Naylor, B. A., 2010, "OpenMDAO: An Open Source Framework for Multidisciplinary Analysis and Optimization," AIAA/ISSMO Multidisciplinary Analysis Optimization Conference Proceedings.
[3] Hashemi, A., 2008, "Integrated Multidisciplinary Modeling, Virtual Design and Innovation," Proceedings of Advanced Technology Center Colloquium, Lockheed Martin.
[4] Cramer, E. J., 2010, "MDO is a State of Mind," Proceedings of National Science Foundation Workshop on the Future of Multidisciplinary Design Optimization: Advancing the Design of Complex Systems, National Science Foundation.
[5] Hazelrigg, G. A., 2003, "Thoughts on Model Validation for Engineering Design," *Proceedings of the ASME Design Engineering Technical Conference*, Vol. 3, pp. 373–380.
[6] Davis, M. D., Signal, R., and Weyuker, E. J., 1994, *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, Morgan Kauffman, San Francisco, CA.
[7] Cellier, F. E., and Koffman, E., 2006, *Continuous System Simulation*, Springer, New York, NY.
[8] Meyer, A., 2004, "Methods for Determining the Validity of a Model," *Principles of Functional Verification*, Newnes, Burlington, pp. 21–48.
[9] Abadi, M., and Lamport, L., 1993, "Composing Specifications," ACM Trans. Program. Lang. Syst., **15**, p. 73.
[10] Segura, S., Hierons, R. M., Benavides, D., and Ruiz-Corts, A., 2011, "Automated Metamorphic Testing on the Analyses of Feature Models," Inf. Softw. Technol., **53**(3), pp. 245–258.
[11] Mosterman, P. J., and Vangheluwe, H., 2004, "Computer Automated Multi-Paradigm Modeling: An Introduction," Simulation, **80**(9), pp. 433–450.
[12] Shaja, A., and Sudhakar, K., 2010, "Optimized Sequencing of Analysis Components in Multidisciplinary Systems," Res. Eng. Des., **21**, pp. 173–187.
[13] Jones, D. R., Schonlau, M., and Welch, W. J., 1998, "Efficient Global Optimization of Expensive Black-Box Functions," J. Global Optim., **13**(4), pp. 455–492.
[14] Kleijnen, J. P. C., and van Beers, W. C. M., 2004, "Application-Driven Sequential Designs for Simulation Experiments: Kriging Metamodelling," J. Oper. Res. Soc., **55**(8), pp. 876–883.

[15] Xiong, Y., Chen, W., and Tsui, K., 2008, "A New Variable-Fidelity Optimization Framework Based on Model Fusion and Objective-Oriented Sequential Sampling," ASME J. Mech. Des., **130**(11), p. 111401.
[16] Shan, S., and Wang, G. G., 2011, "Turning Black-Box Functions into White Functions," ASME J. Mech. Des., **133**(3), p. 031003.
[17] Huang, Y., and Chan, K., 2010, "A Modified Efficient Global Optimization Algorithm for Maximal Reliability in a Probabilistic Constrained Space," ASME J. Mech. Des., **132**(6), p. 061002.
[18] Devanathan, S., and Ramani, K., 2010, "Creating Polytope Representations of Design Spaces for Visual Exploration Using Consistency Techniques," ASME J. Mech. Des., **132**(8), p. 081011.
[19] Malak, R. J., Jr., and Paredis, C. J. J., 2010, "Using Support Vector Machines to Formalize the Valid Input Domain of Predictive Models in Systems Design Problems," ASME J. Mech. Des., **132**(10), p. 101001.
[20] Sosa, M. E., Eppinger, S. D., and Rowles, C. M., 2007, "A Network Approach to Define Modularity of Components in Complex Products," ASME J. Mech. Des., **129**(11), pp. 1118–1129.
[21] Larson, B., and Mattson, C. A., 2010, "Developing System Behavioral Models by Integrating Discipline Specific Models," Proceedings of Structures, Structural Dynamics, and Materials Conference Proceedings, AIAA.
[22] Zeigler, B. P., Praehofer, H., and Kim, T. G., 2000, *Theory of Modeling and Simulation*, 2nd ed., Academic Press, San Diego, CA.
[23] Khalil, H. K., 1996, *Nonlinear Systems*, 2nd ed., Prentice Hall, Upper Saddle River, NJ.
[24] IEEE, S. I. S. C., 2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Framework and Rules," IEEE, New York, NY.
[25] Oses, N., Pidd, M., and Brooks, R. J., 2004, "Critical Issues in the Development of Component Based Discrete Simulation," Simul. Model. Pract. Theory, **121**(7–8), pp. 495–514.
[26] Emanuelsson, P., and Nilsson, U., 2008, "A Comparative Study of Industrial Static Analysis Tools," Electron. Notes Theor. Comput. Sci., **217**, pp. 5–21 (Proceedings of the 3rd International Workshop on Systems Software Verification (SSV 2008)).
[27] Larson, B. J., Anderson, T. V., and Mattson, C. A., 2010, "System Behavioral Model Accuracy for Concurrent Design and Modeling," Proceedings of 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference Proceedings, AIAA.
[28] Mckay, M. D., Beckman, R. J., and Conover, W. J., 2000, "Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code," Technometrics, **42**(1), pp. 55–61.
[29] Isaaks, E. H., and Srivastava, R. M., 1989, *Applied Geostatistics*, Oxford University Press, New York.
[30] Lophaven, S. N., Nielsen, H. B., and Sondergaard, J., 2002, "DACE, a Matlab Kriging Toolbox," Technical University of Denmark, Kgs. Lyngby, Denmark, Technical Report No. IMM-TR-2002-12.
[31] Gonzalez, R. C., and Woods, R. E., 2008, *Digital Image Processing*, 3rd ed., Prentice-Hall, Englewood Cliffs, NJ.