ASME Journal of Mechanical Design
Online journal at:
https://asmedigitalcollection.asme.org/mechanicaldesign

A GPU-Based Parallel Region Classification Method for Continuous Constraint Satisfaction Problems

Guanglu Zhang
Mem. ASME
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: glzhang@cmu.edu

Wangchuan Feng
Mem. ASME
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213

Jonathan Cagan
Fellow ASME
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: cagan@cmu.edu

Continuous constraint satisfaction is prevalent in many science and engineering fields. When solving continuous constraint satisfaction problems, it is more advantageous for practitioners to derive all feasible regions (i.e., the solution space) rather than a limited number of solution points, since these feasible regions facilitate design concept generation and design tradeoff evaluation. Several central processing unit (CPU)-based branch-and-prune methods and geometric approximation methods have been proposed in prior research to derive feasible regions for continuous constraint satisfaction problems. However, these methods have not been extensively adopted in practice, mainly because of their high computational expense. To overcome the computational bottleneck of extant CPU-based methods, this paper introduces a GPU-based parallel region classification method to derive feasible regions for continuous constraint satisfaction problems in a reasonable computational time. Using interval arithmetic, coupled with the computational power of GPU, this method iteratively partitions the design space into many subregions and classifies these subregions as feasible, infeasible, and indeterminate regions. To visualize these classified regions in the design space, a planar visualization approach that projects all classified regions into one figure is also proposed. The GPU-based parallel region classification method and the planar visualization approach are validated through two case studies about the bird function and the welded beam design. These case studies show that the method and the approach can solve the continuous constraint satisfaction problems and visualize the results effectively and efficiently. A four-step procedure for implementing the method and the approach in practice is also outlined. [DOI: 10.1115/1.4063158]

Keywords: constraint satisfaction, design optimization, engineering design, GPU programming, interval arithmetic, parallel computing, design automation, design evaluation, design visualization, multidisciplinary design and optimization, multi-objective optimization

1 Introduction

A continuous constraint satisfaction problem, also known as a numerical constraint satisfaction problem, is defined by sets of variable values that must satisfy a finite number of constraints, where each variable ranges over a continuous domain. Continuous constraint satisfaction has significant applications in many science and engineering fields, such as material science, chemical engineering, and mechanical engineering [1–4]. For example, material scientists need to assess the manufacturing feasibility of a material based on constraints determined by the material composition and its processing conditions (e.g., pressure and temperature) [5]; mechanical engineers are required to meet several constraints related to system and component performances (e.g., speed and payload) when they design an unmanned aerial vehicle [6].

For a continuous constraint satisfaction problem, the allowable ranges of variables define a design space. Each solution of the problem (i.e., a set of variable values that satisfy all specified constraints) corresponds to a point in the design space, referred to as a solution point. In practice, sampling-based methods [7,8] have been commonly employed to derive multiple solution points for continuous constraint satisfaction problems. However, in contrast to solution points, when the closed-form expression of each constraint is available, it is more advantageous for practitioners to derive all feasible regions, also known as solution regions or solution space, in the design space, where any point within a feasible region satisfies all specified constraints. Importantly, unlike a limited number of solution points, these feasible regions reveal all possible locations in the design space where solutions of a continuous constraint satisfaction problem could exist. Therefore, these feasible regions allow practitioners to comprehensively consider design factors that cannot be formally expressed and evaluate design tradeoffs in their design process [9,10]. In addition, these feasible regions also facilitate the negotiation process during multi-party collaboration [11].

Manuscript received March 17, 2023; final manuscript received July 31, 2023; published online December 6, 2023. Assoc. Editor: Douglas Allaire.

To derive the feasible regions for continuous constraint satisfaction problems, several branch-and-prune methods [12–15] and geometric approximation methods [9,16–18] have been proposed in prior research, but these methods have not been extensively adopted in practice, mainly because of their high computational expense. Although the recent advances in the GPU offer new opportunities to overcome the computational bottleneck [19], extant branch-and-prune methods and geometric approximation methods cannot utilize the computational power of GPU because these methods are designed for central processing unit (CPU)-based sequential computing.

Moreover, extant branch-and-prune methods and geometric approximation methods output sets of numerical values that represent the locations of feasible regions in the design space. In contrast to these numerical values, a visual representation of all feasible regions in the design space is more helpful for practitioners to identify solutions of the problem and make informed decisions accordingly. However, when a continuous constraint satisfaction problem includes more than three variables (i.e., $n > 3$), the feasible regions derived from these extant methods are usually visualized through multiple two-dimensional or three-dimensional figures. Practitioners have to jump back and forth between these figures and cannot visually observe all n-dimensional feasible regions in one figure.

In this paper, a GPU-based parallel region classification method is introduced to derive the feasible regions for continuous constraint satisfaction problems. The method applies to a finite number of inequality constraints that have closed-form expressions. Using interval arithmetic, coupled with the computational power of GPU, this method iteratively partitions the design space into many subregions and classifies these subregions as feasible, infeasible, and indeterminate regions, where any point within an infeasible region cannot satisfy one or more specified constraints. Specifically, at each iteration step, the indeterminate region(s) classified in the last iteration step are partitioned into many subregions based on the compute capability of the practitioner's GPU, and these subregions are evaluated using interval arithmetic and classified based on the specified constraints in parallel on the GPU. Besides the GPU-based parallel region classification method, a planar visualization approach that projects all n-dimensional feasible regions in the design space into one figure is also proposed to enable more effective solution identification for continuous constraint satisfaction problems.

This paper begins with a brief review of extant methods to solve continuous constraint satisfaction problems and the background knowledge of interval arithmetic in Sec. 2. Section 3 presents the GPU-based parallel region classification method and the planar visualization approach for continuous constraint satisfaction problems. Four steps for implementing the GPU-based parallel region classification method and the planar visualization approach in practice are provided in Sec. 4. The application of the method and the approach is demonstrated through two case studies of the bird function and the welded beam design in Sec. 5. The paper concludes with a discussion of the contribution of the work and future research directions.

2 Background and Related Work

This section briefly reviews extant methods for continuous constraint satisfaction, with a focus on the engineering design context. A thorough review of constraint satisfaction, where variables are defined in continuous and discrete domains, can be found in books and review papers written or edited by Apt [20], Tsang [21], Rossi et al. [22], Kumar [23], and Brailsford et al. [24], among others. The background knowledge of interval arithmetic is also provided in this section.

2.1 Extant Methods for Continuous Constraint Satisfaction.

The goal of continuous constraint satisfaction is to find sets of variable values that satisfy all specified constraints. Each set of variable values that satisfy all specified constraints corresponds to a solution point in the design space. Sampling-based methods, also known as designs for computational experiments [25–27], such as maximum entropy designs, mean squared-error designs, Latin hypercubes, and randomized orthogonal arrays, are commonly employed to derive multiple solution points for continuous constraint satisfaction problems [7,8]. Using these sampling-based methods, a finite number of sample points are first chosen from the design space. The corresponding variable values at each sample point are then plugged into the specified constraints to filter out the points at which one or multiple constraints cannot be satisfied, and the remaining points are the solution points. These sampling-based methods provide a straightforward way to explore the design space, but these methods are not able to delineate feasible regions from the design space, where any solution point within a feasible region satisfies all specified constraints. Since several design factors (e.g., aesthetic and socio-economic factors) often cannot be formally expressed in practice, practitioners only can consider these factors at a limited number of solution points during their design process when they employ these sampling-based methods to solve continuous constraint satisfaction problems [10]. Moreover, practitioners cannot comprehensively evaluate design tradeoffs (e.g., the impact of adding a new constraint or removing an existing constraint) based on the solution points derived by sampling-based methods [9].

When the closed-form expression of each constraint is available, several branch-and-prune methods [12–15] and geometric approximation methods [9,16–18] have been proposed to delineate feasible regions from the design space for continuous constraint satisfaction problems. These branch-and-prune methods aim to generate a set of n-dimensional hyperrectangular regions, also known as boxes, that enclose all feasible regions in the design space. A typical branch-and-prune method relies on an iterative process that consists of many branching steps and pruning steps, where a branching step usually partitions a chosen region into two or more subregions in a specific dimension in the design space (i.e., only partition the range of one specific variable in the chosen region), and a pruning step reduces the size of a chosen region by eliminating infeasible subregions within the region [12,13]. Commonly used techniques in the pruning step include but are not limited to hull consistency, box consistency, constraint inversion, and dichotomous search [12,28,29]. The iterative process is halted when a user-specified region size tolerance is reached. In contrast, extant geometric approximation methods endeavor to represent feasible regions in the design space using many three-dimensional geometric elements, such as orthogonal polyhedra [17], parallelepipeds [18], and polytopes [9]. For example, a polytope representation method introduced by Devanathan and Ramani [9] includes the following four steps:

(1) Formulate the continuous constraint satisfaction problem;
(2) Transform all constraints into ternary constraints;
(3) Create the feasible regions for each ternary constraint separately;
(4) Prune these feasible regions using consistency to obtain the feasible regions for all ternary constraints.

As demonstrated by the four-step procedure above, extant geometric approximation methods are only able to address ternary constraints, where each constraint includes three variables. For constraints involving more than three variables, practitioners must introduce new variables (i.e., auxiliary variables) and rewrite each of these constraints as multiple ternary constraints [16,30].

Despite several successful applications of these branch-and-prune methods and geometric approximation methods, sampling-based methods are still commonly used in the engineering design process [6,31]. A major reason is that these branch-and-prune methods and geometric approximation methods are significantly more computationally expensive than sampling-based methods. Specifically, extant branch-and-prune methods usually take many

iteration steps to reach the user-specified region size tolerance because each branching step usually only partitions the chosen region in one dimension (e.g., bisect the range of one variable and keep intact the ranges of other variables in the chosen region) [12]; extant geometric approximation methods often require very high resolution (i.e., the size of each geometric element must be very small) to avoid erroneous approximation of the feasible regions for all ternary constraints [9]. Notably, the recent advances in GPU offer new opportunities to overcome the computational bottleneck [19]. With the falling price of GPU, many personal laptops and desktops are now equipped with a dedicated GPU. However, since GPU is designed for *parallel computing*, extant branch-and-prune methods and geometric approximation methods based on CPU *sequential computing* cannot utilize the computational power of GPU.

In addition, the result output from extant branch-and-prune methods and geometric approximation methods is sets of numerical values that represent the locations of feasible regions. In practice, a visual representation of all feasible regions in the design space is helpful for practitioners to identify solutions of the problem and make informed decisions accordingly. However, when a continuous constraint satisfaction problem includes more than three variables (i.e., $n>3$), the feasible regions derived from these extant methods are usually visualized through multiple two-dimensional or three-dimensional figures. For example, a total of ten three-dimensional figures are necessary to visualize feasible regions when a constraint satisfaction problem includes five variables [15]. As a result, practitioners have to jump back and forth between these figures when they generate or evaluate design concepts based on feasible regions in the n-dimensional design space.

2.2 Interval Arithmetic. To overcome the computational bottleneck discussed in Sec. 2.1, a GPU-based parallel region classification method is introduced in this paper for continuous constraint satisfaction problems based on the interval arithmetic. Interval arithmetic, also known as interval analysis or interval computation, is a mathematical technique designed to automatically provide rigorous bounds on rounding errors, approximation errors, and propagated uncertainties in mathematical computation [32,33]. Unlike floating-point computations where each value is represented as a number (e.g., 2.01), interval arithmetic represents each value as a range (e.g., [1.95, 2.11]). A list of interval arithmetic operations has been specified in IEEE 1788 standard [34], and four basic interval arithmetic operations (i.e., addition, subtraction, multiplication, and division) are provided in Eqs. (1)–(4) for demonstration purposes [35], where an interval $[a, b]$ is represented in square brackets, a is the lower bound, and b is the upper bound of the interval:

$$[x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2] \tag{1}$$

$$[x_1, x_2] - [y_1, y_2] = [x_1 - y_2, x_2 - y_1] \tag{2}$$

$$[x_1, x_2] \cdot [y_1, y_2] = \begin{bmatrix} \min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, \\ \max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\} \end{bmatrix} \tag{3}$$

$$\frac{[x_1, x_2]}{[y_1, y_2]} = [x_1, x_2] \cdot \frac{1}{[y_1, y_2]} \tag{4}$$

In practice, since each numerical value is approximated to its closest binary machine number and stored in computers [36], outward rounding is implemented for every interval arithmetic operation to provide rigorous bounds for the solution [37]. The outwardly rounded lower bound (left endpoint) is the closest machine number less than or equal to the exact lower bound, and the outwardly rounded upper bound (right endpoint) is the closest machine number greater than or equal to the exact upper bound [32]. For example, the interval solution [1.9986, 2.0014] could be outwardly rounded as [1.998, 2.002] with four significant digits.

Although the outwardly rounded upper and lower bounds derived by interval arithmetic operations have been proved to create an interval that contains all possible solutions (i.e., rigorous bounds for the solution) [32,33], many interval computations including multiple interval arithmetic operations suffer the dependence problem, also known as interval dependency. The dependence problem results in overestimated bounds for the solution (i.e., an interval with unwanted excess width). For example, for the function $f(x) = x - x^2$, where $x \in [0, 1]$, the exact range of the function value is [0, 0.25]; the interval arithmetic operations shown in Eqs. (2) and (3) give $[0, 1] - [0, 1] \cdot [0, 1] = [-1, 1]$, where the result $[-1, 1]$ includes the exact range [0, 0.25], but it has significant excess width. Such overestimated bounds are produced because each occurrence of the variable x is treated as a different variable in the interval computation. Notably, if every variable occurs only once in a function, the interval evaluation of the function gives the exact range of the function value. However, when any variable occurs more than once in a function, the dependence problem often appears. In practice, an interval computation including multiple interval arithmetic operations is often expressed as a sequence of operations, commonly known as a *code list*, to facilitate computer programming [32]. The dependence problem could be identified by counting the number of times each variable occurs in a code list. For example, a function is defined as:

$$g(x_1, x_2) = e^{x_1 + x_2^2} - x_1 \cdot x_2 \tag{5}$$

The interval evaluation of the function defined by Eq. (5) is expressed as a code list:

$$T_1 = x_2^2 \tag{6}$$

$$T_2 = x_1 + T_1 \tag{7}$$

$$T_3 = e^{T_2} \tag{8}$$

$$T_4 = x_1 \cdot x_2 \tag{9}$$

$$g(x_1, x_2) = T_3 - T_4 \tag{10}$$

Since both variables x_1 and x_2 occur more than once in the code list, the dependence problem appears when the code list represented by Eqs. (6)–(10) is employed to compute the bounds of the function value based on interval arithmetic.

The dependence problem is a major factor that impedes the application of interval arithmetic. Several approaches have been introduced by researchers to produce exact bounds or at least narrower bounds for interval computation [28,32,38–40]. One major approach is to rewrite the function in another form [28,32], such as a form where every variable only occurs once (if possible), the centered form, the mean value form, or a Taylor series. In the first example, the function could be rewritten as $f(x) = x - x^2 = -(x - 0.5)^2 + 0.25$, and the interval evaluation of the rewritten function leads to the exact range [0, 0.25]. Another approach, known as splitting, is to divide the range of each variable into d subintervals and then take the union of the d interval evaluations over the elements of the subdivision. It has been proved that the union of the d interval evaluations converges to the exact range of the function value when d approaches ∞ [32]. In the first example, when the range $x \in [0, 1]$ is uniformly divided into 1000 subintervals, the union of the 1000 interval evaluations of the function $f(x) = x - x^2$ in each subinterval gives $[-0.001, 0.251]$, which is close to the exact range [0, 0.25] [32].

3 Region Classification and Visualization for Continuous Constraint Satisfaction

Based on the research gap discussed in Sec. 2.1, a GPU-based parallel region classification method is introduced in this section to

derive feasible regions in the design space for continuous constraint satisfaction problems. Unlike extant CPU-based branch-and-prune methods and geometric approximation methods that only evaluate one region in the design space at a time, at each iteration step, the method introduced in this section partitions indeterminate region(s) in the design space into many subregions based on the compute capability of the practitioner's GPU and evaluates these subregions in parallel on the GPU. Such GPU-based parallel computing significantly improves problem-solving efficiency. Moreover, the method is able to solve the continuous constraint satisfaction problem involving more than three variables, and practitioners do not need to transform all constraints into ternary constraints when they implement the method. Besides the GPU-based parallel region classification method, a planar visualization approach is also introduced in this section to project all n-dimensional feasible regions in the design space into one figure for continuous constraint satisfaction problems. Practitioners can observe the location of each feasible region in the design space from the figure and make informed decisions accordingly.

3.1 The GPU-Based Parallel Region Classification Method. For a continuous constraint satisfaction problem with a finite set of inequality constraints $\boldsymbol{C} = \{c_1, c_2, \ldots, c_m\}$, these inequality constraints are defined as

$$\{c_j : g_j(x_1, x_2, \ldots, x_n) \leq 0\} \tag{11}$$

where $j \in \{1, 2, \ldots, m\}$. There are a set of n variables $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$, and these variables range over continuous domains $\boldsymbol{D} = \{d_1, d_2, \ldots, d_n\}$ defined as

$$\{d_i : x_i \in [x_i^{lb}, x_i^{ub}]\} \tag{12}$$

where $i \in \{1, 2, \ldots, n\}$, and x_i^{lb} and x_i^{ub} are the lower and upper bounds of variable x_i, respectively. The objective is to find sets of n variable values that satisfy all the constraints $\boldsymbol{C} = \{c_1, c_2, \ldots, c_m\}$. Notably, $g_j(x_1, x_2, \ldots, x_n)$ in Eq. (11) represents the closed-form expression of an inequality constraint function. Solving the continuous constraint satisfaction problems where constraints do not have closed-form expressions is beyond the scope of this paper. Equality constraints are also not included in the continuous constraint satisfaction problem defined by Eqs. (11) and (12). If feasible, each equality constraint could be employed to eliminate a variable from a continuous constraint satisfaction problem in practice. An equality constraint also could be replaced by two inequality constraints that are close to each other when such an approximation is acceptable [16]. For example, an equality constraint $f(\boldsymbol{X}) = 0$ may be replaced by two inequality constraints $f(\boldsymbol{X}) - \varepsilon \leq 0$ and $-f(\boldsymbol{X}) - \varepsilon \leq 0$ (i.e., $-\varepsilon \leq f(\boldsymbol{X}) \leq \varepsilon$), where ε is a user-specified small positive value.

To solve the continuous constraint satisfaction problem, the n-dimensional design space established by Eq. (12) is iteratively partitioned into many subregions, and these subregions are classified as feasible, infeasible, and indeterminate regions, where these three types of regions are defined as:

Feasible region: if any point within the region satisfies all the constraints $\boldsymbol{C} = \{c_1, c_2, \ldots, c_m\}$.

Infeasible region: if any point within the region cannot satisfy one or multiple constraints in $\boldsymbol{C} = \{c_1, c_2, \ldots, c_m\}$.

Indeterminate region: if points within the region may or may not satisfy all the constraints $\boldsymbol{C} = \{c_1, c_2, \ldots, c_m\}$.

The whole design space is first evaluated using interval arithmetic. Each constraint function $g_j(\boldsymbol{X})$ is expressed as a code list, as demonstrated in Sec. 2.2. The allowable ranges of n variables defined by Eq. (12) are then plugged into each code list, and the sequence of interval arithmetic operations in each code list yields $[\mathrm{DLB}_j, \mathrm{DUB}_j]$, where DLB_j and DUB_j are the lower and upper bounds for the value of the constraint function $g_j(\boldsymbol{X})$ within the design space, respectively. Notably, the interval bounds derived by each interval arithmetic operation are outwardly rounded to the user-specified digits or significant digits (e.g., eight significant digits), and the interval $[\mathrm{DLB}_j, \mathrm{DUB}_j]$ is therefore guaranteed to include all possible values of $g_j(\boldsymbol{X})$ within the design space. If any variable occurs more than once in a code list, the dependency problem needs to be considered, and the function rewriting approach and/or the splitting approach discussed in Sec. 2.2 could be used to derive the exact or narrower bounds for the value of the constraint function $g_j(\boldsymbol{X})$ within the design space.

Based on the lower and upper bounds for the value of each constraint function, DLB_j and DUB_j, and the inequality defined by Eq. (11), the whole design space is classified as a feasible region, an infeasible region, or an indeterminate region. Specifically, if $\mathrm{DUB}_j \leq 0$ for all $j \in \{1, 2, \ldots, m\}$, the design space is a feasible region; if $\mathrm{DLB}_j > 0$ for any $j \in \{1, 2, \ldots, m\}$, the design space is an infeasible region; otherwise, the design space is an indeterminate region. When the whole design space is classified as a feasible region or an infeasible region, practitioners need to check each constraint and consider tightening or relaxing certain constraint(s) if necessary.

When the whole design space is classified as an indeterminate region, the design space is iteratively partitioned into subregions. At each iteration step, a GPU-based parallel bound-and-classify process is carried out to classify these subregions as feasible regions, infeasible regions, and indeterminate regions. Only the indeterminate regions are further partitioned in the next iteration step. Since it is not necessary to further partition the feasible and infeasible regions in the next iteration step, the number of subregions will likely not increase exponentially in the iteration process. The iteration process is halted when the user-specified stopping criteria are satisfied. The partition strategy, the GPU-based parallel bound-and-classify process, and the stopping criteria are discussed as follows.

At the beginning of each iteration step, the indeterminate region(s) classified in the last iteration step are partitioned into many subregions. Each indeterminate region is partitioned in multiple dimensions at the same time using a user-specified partition method, such as uniform partition and golden ratio partition. Different scales and coordinate systems also could be employed if necessary. For example, a logarithmic scale could be applied to a variable if the variable is defined over a large range (e.g., $[10^{-6}, 10^{6}]$). The total number of subregions that results from the partition are chosen based on the compute capability of the practitioner's GPU. For example, at each iteration step, each of the indeterminate regions could be partitioned in all its n dimensions at the same time. In each dimension, the interval of the corresponding variable is uniformly partitioned into k subintervals, where the integer k is estimated through

$$k \approx \sqrt[n]{\frac{N_{\mathrm{GPU}}}{N_{\mathrm{idt}}}} \text{ when } N_{\mathrm{idt}} \cdot 2^n < N_{\mathrm{GPU}} \tag{13}$$

$$k = 2 \text{ when } N_{\mathrm{idt}} \cdot 2^n \geq N_{\mathrm{GPU}} \tag{14}$$

where N_{GPU} is the number of shading units in the practitioner's GPU, and N_{idt} is the number of indeterminate regions that need to be partitioned at the iteration step. Notably, the partition strategy defined by Eqs. (13) and (14) is presented here for demonstration purposes, and practitioners are recommended to customize the partition strategy based on their specific problem when they implement the method. The partition strategy defined by Eqs. (13) and (14) avoids GPU underutilization in the first few iteration steps when N_{idt} has a small value (e.g., $N_{\mathrm{idt}} = 1$ in the first iteration step), but this partition strategy can lead to a significant number of subregions in an iteration step when many indeterminate regions need to be partitioned in the iteration step (e.g., $N_{\mathrm{idt}} > 10^6$) or the continuous constraint satisfaction problem includes many variables (e.g., $n > 20$).

The subregions that result from the partition are then evaluated and classified as feasible, infeasible, and indeterminate regions in parallel on the GPU during each iteration step (i.e., a GPU-based parallel bound-and-classify process during an iteration step). Based on the code list of each constraint function $g_j(\boldsymbol{X})$, the interval

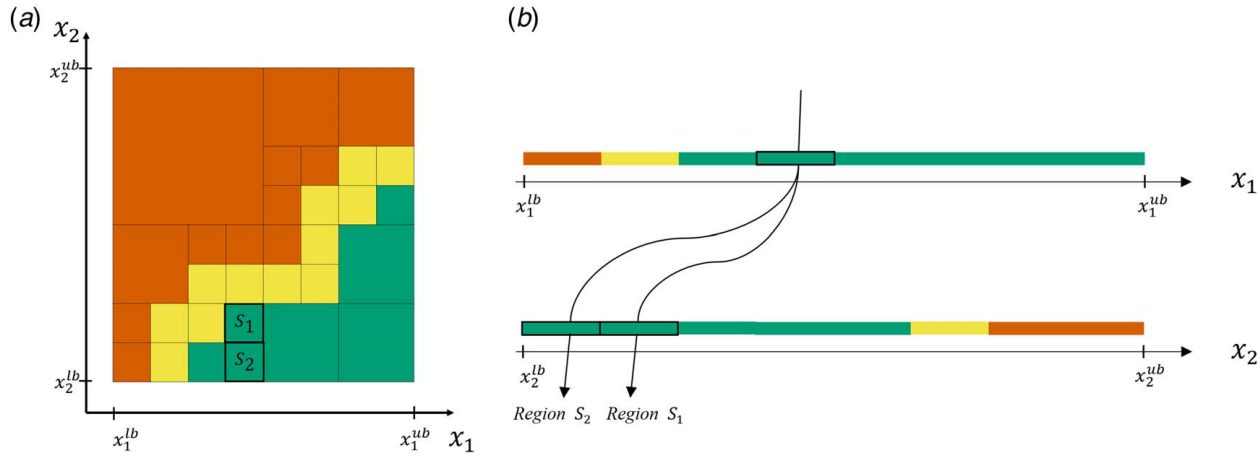

Fig. 1 An illustration of the planar visualization approach: (*a*) *n*-dimensional infeasible (red), indeterminate (yellow), and feasible (green) regions, and (*b*) projection from *n*-dimensional regions to one figure with *n* horizontal axes

evaluation of each constraint function within each subregion gives $[LB_j^{(s)}, UB_j^{(s)}]$. Here, $LB_j^{(s)}$ and $UB_j^{(s)}$ are the outwardly rounded lower and upper bounds for the value of the constraint function $g_j(\boldsymbol{X})$, respectively, s is the number of the subregion, and $s \in \{1, 2, \ldots, N_s\}$, where N_s is the total number of subregions that need to be evaluated at the iteration step. The subregion s is classified as a feasible region if $UB_j^{(s)} \leq 0$ for all $j \in \{1, 2, \ldots, m\}$. If $LB_j^{(s)} > 0$ for any $j \in \{1, 2, \ldots, m\}$, the subregion s is classified as an infeasible region. If the subregion s is not a feasible or infeasible region, it is classified as an indeterminate region. Notably, when any variable occurs more than once in a code list, practitioners are recommended to use the function rewriting approach and/or the splitting approach discussed in Sec. 2.2 to derive the exact or narrower bounds for the value of the constraint function $g_j(\boldsymbol{X})$ within each subregion (i.e., reduce excess width for $[LB_j^{(s)}, UB_j^{(s)}]$) at least during the first few iteration steps. Using the exact or narrower bounds, more subregions could be classified as feasible and infeasible regions in an iteration step, and thus less indeterminate regions need to be partitioned in the next iteration step.

At the end of each iteration step, if the user-specified stopping criteria are satisfied, the iteration process is halted. Otherwise, a new iteration step begins, and the indeterminate region(s) classified in the last iteration step are further partitioned. Practitioners can specify one or multiple stopping criteria based on computational power and *n*-dimensional region volume when they implement the method. The stopping criteria related to computational power include but are not limited to the computer run time, the maximum number of iteration steps, the maximum number of subintervals over the range of each variable, and the region size tolerance (e.g., the smallest region width in any specific dimension or all dimensions). The stopping criteria related to *n*-dimensional region volume could be defined using the increment of the *n*-dimensional volume of all classified feasible regions between two adjacent iteration steps or the *n*-dimensional volume of all classified indeterminate regions at the end of an iteration step. Notably, since a user-specified stopping criterion related to *n*-dimensional region volume may take an unaffordable computer run time to be satisfied (or even may not be satisfied), at least one stopping criterion related to computational power should be employed in practice to halt the iteration process in due time. For example, practitioners could specify the maximum number of iteration steps and a limit value for the *n*-dimensional volume of indeterminate regions as two stopping criteria when they implement the method, and the iteration process is halted if any of these two stopping criteria is satisfied.

When the iteration process is halted, all feasible, infeasible, and indeterminate regions classified during the iteration process are output as sets of intervals, where each interval set includes *n* intervals that correspond to one classified region in the design space. The union of all feasible regions creates a *sound* representation of the solution space for the continuous constraint satisfaction problem defined by Eqs. (11) and (12) since any point within these feasible regions satisfies all specified constraints. Meanwhile, the union of all feasible and indeterminate regions creates a *complete* representation of the solution space for the problem since all solution points are guaranteed to be included in these feasible and indeterminate regions. Notably, points within indeterminate regions may or may not satisfy all specified constraints of the problem.

3.2 The Planar Visualization Approach. All *n*-dimensional feasible, infeasible, and indeterminate regions classified during the iteration process are projected into one figure using a planar visualization approach illustrated in Fig. 1. The *n*-dimensional regions classified by the method are shown in Fig. 1(*a*), where red, yellow, and green regions represent infeasible, indeterminate, and feasible regions, respectively. There are two variables in Fig. 1 (i.e., $n = 2$) for a clear illustration of the projection process, and the planar visualization approach is applicable to problems with higher dimensions as demonstrated in the case study in Sec. 5.2, where the classified regions cannot be directly visualized through a two- or three-dimensional figure. The classified regions shown in Fig. 1(*a*) are projected into one figure with *n* horizontal axes for planar visualization, as shown in Fig. 1(*b*). Each horizontal axis in Fig. 1(*b*) corresponds to one variable. The allowable range of each variable defined by Eq. (12) is paved with red, yellow, and green blocks in Fig. 1(*b*). The interpretations of these three types of blocks are presented as follows.

The length of a red block in the horizontal axis defines a specific range for the corresponding variable, where any variable value that belongs to the specific range cannot satisfy one or multiple constraints defined by Eq. (11) *regardless of* the values of other variables. The projection from *n*-dimensional infeasible regions to red blocks for planar visualization is illustrated in Fig. 7 in Appendix A.

The length of a yellow block in the horizontal axis defines a specific range for the corresponding variable, where any variable value that belongs to the specific range may or may not satisfy all the constraints defined by Eq. (11) *regardless of* the values of other variables. The projection from *n*-dimensional indeterminate regions to yellow blocks for planar visualization is illustrated in Fig. 8 in Appendix A.

The length of a green block in the horizontal axis defines a specific range for the corresponding variable, where any variable value that belongs to the specific range satisfies all the constraints defined by Eq. (11) *only if* the values of other variables belong to the ranges defined by the corresponding feasible region(s). The projection from *n*-dimensional feasible regions to green blocks for planar visualization is illustrated in Fig. 9 in Appendix A.

The planar visualization approach projects all classified *n*-dimensional regions into one figure, and practitioners can observe

the ranges for variables where the solution for the continuous constraint satisfaction problem could exist from the green blocks in the figure. It is of note that user interaction is required to retrieve the information of specific feasible region(s) (i.e., sets of intervals) from the figure. For example, once practitioners specify a specific range for a variable within a green block, the ranges of other variables that lead to the feasible region(s) in the design space could be highlighted and connected by arrows in the planar visualization figure, as illustrated by region S_1 and region S_2 in Fig. 1(*b*).

4 Four Steps to Solve Continuous Constraint Satisfaction Problems

A GPU-based parallel region classification method is introduced in Sec. 3 for continuous constraint satisfaction problems. The method iteratively partitions the design space into many subregions and classifies these subregions as feasible, infeasible, and indeterminate regions. The classified regions are then projected into one figure using a planar visualization approach. In this section, the procedure to implement the GPU-based parallel region classification method and the planar visualization approach is summarized in four steps. These four steps are illustrated in Fig. 2. Practitioners can follow these steps to solve continuous constraint satisfaction problems and visualize their results.

Step 1—Problem definition: A continuous constraint satisfaction problem is defined and formulated based on Eqs. (11) and (12). The allowable ranges of n variables define an n-dimensional design space. All constraints are formulated as inequality constraints with closed-form expressions. If equality constraint(s) exist, each equality constraint could be employed to eliminate a variable from the problem or replaced by two inequality constraints that are close to each other when such an approximation is acceptable.

Step 2—Initial evaluation and classification of the whole design space: The constraint functions are evaluated within the whole design space using interval arithmetic. Based on the interval evaluations of all constraint functions, the whole design space is classified as a feasible, infeasible, or indeterminate region. If the whole design space is classified as a feasible or infeasible region, practitioners need to check each constraint and consider tightening or relaxing certain constraint(s) if necessary.

Step 3—Iterative partition and classification of the design space: If the whole design space is classified as an indeterminate region, the design space is iteratively partitioned into many subregions, and these subregions are classified as feasible regions, infeasible regions, and indeterminate regions. Specifically, at each iteration step, the indeterminate region(s) classified in the last iteration step are partitioned into many subregions using a user-specified partition strategy. The constraint functions are then evaluated within these subregions using interval arithmetic in parallel on a GPU. Each subregion is classified as a feasible, infeasible, or indeterminate region based on the interval evaluations of all constraint functions within the subregion. The iteration process is halted when the user-specified stopping criteria are satisfied.

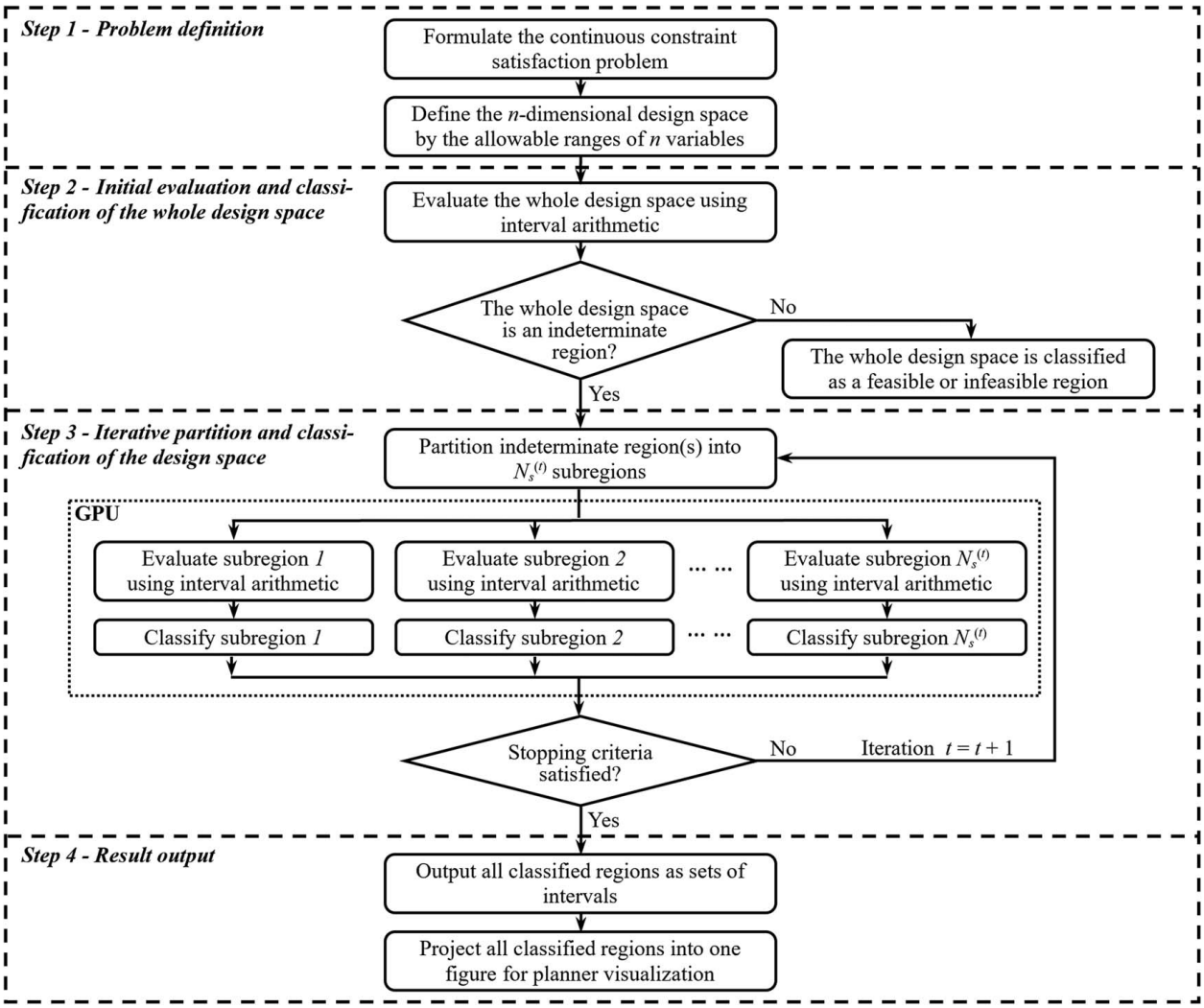

Fig. 2 Four steps to apply the GPU-based parallel region classification method and the planar visualization approach

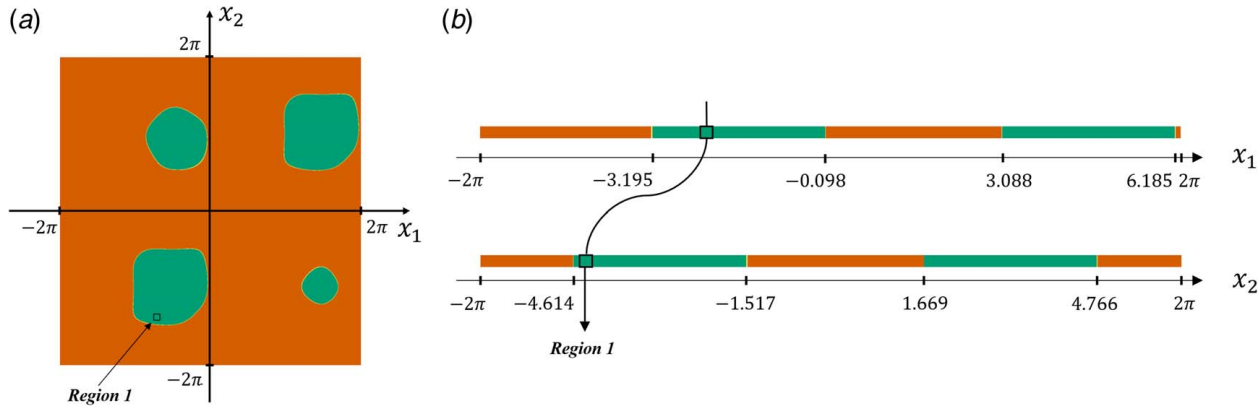

Fig. 3 Planar visualization for the result of the bird function case study: (*a*) two-dimensional infeasible (red), indeterminate (yellow), and feasible (green) regions, and (*b*) projection from two-dimensional regions to one figure with two horizontal axes

Step 4—Result output: When the iteration process is halted, all feasible, infeasible, and indeterminate regions classified during the iteration process are output as sets of intervals, where each interval set corresponds to one classified region in the design space. Besides these sets of intervals, all classified regions are also projected into one figure for planar visualization. The green blocks in the figure indicate the ranges for variables where the solution for the continuous constraint satisfaction problem could exist.

5 Case Studies of the Bird Function and the Welded Beam Design

To demonstrate the four-step procedure outlined in Sec. 4, two continuous constraint satisfaction problems about the bird function and the welded beam design are employed as two case studies in this section. The bird function and the welded beam design are two benchmark optimization problems. In this section, these two problems are explored as continuous constraint satisfaction problems. The results of these case studies validate the effectiveness and efficiency of the GPU-based parallel region classification method and the planar visualization approach. Using the method and the approach, the feasible regions of each problem are derived successfully as sets of intervals, and these feasible regions are also projected into one figure for planar visualization. The results output from the method and the approach facilitate design concept generation and evaluation. The source code to implement the GPU-based parallel region classification method for these two case studies is included in the public repository.[1]

5.1 The Case Study of Bird Function. The bird function is commonly used as a benchmark function to validate the performance of optimization algorithms [41]. The bird function is employed here to define a continuous constraint satisfaction problem including two variables, $\boldsymbol{X} = \{x_1, x_2\}$. The allowable ranges of these two variables, $\boldsymbol{D} = \{d_1, d_2\}$, are specified as

$$\{d_1 : x_1 \in [-2\pi, 2\pi]\} \tag{15}$$

$$\{d_2 : x_2 \in [-2\pi, 2\pi]\} \tag{16}$$

The constraint of the problem, $\boldsymbol{C} = \{c_1\}$, is defined by the bird function as

$$\{c_1 : g_1(\boldsymbol{X}) = (x_1 - x_2)^2 + \sin(x_1) \cdot e^{[1-\cos(x_2)]^2} + \cos(x_2) \cdot e^{[1-\sin(x_1)]^2} \leq 0\} \tag{17}$$

To derive the feasible regions of the continuous constraint satisfaction problem defined by Eqs. (15)–(17), the GPU-based parallel region classification method presented in Secs. 3 and 4 is employed. The allowable ranges of the two variables given by Eqs. (15) and (16) define a two-dimensional design space for the problem. The whole design space is first evaluated using interval arithmetic. The code list of the constraint functions, $g_I(\boldsymbol{X})$, is generated and provided in the public repository.[2] The sequence of interval arithmetic operations in the code list yields the outwardly rounded lower and upper bounds for the value of the constraint function $g_I(\boldsymbol{X})$ within the design space, $[\mathrm{DLB}_I, \mathrm{DUB}_I]$, and the whole design space is classified as an indeterminate region accordingly. The design space is then iteratively partitioned into many subregions, and these subregions are evaluated and classified as feasible regions, infeasible regions, and indeterminate regions in parallel on the GPU of a server at each iteration step, where the server configuration is provided in Appendix B. The partition strategy defined by Eqs. (13) and (14) in Sec. 3 is employed in this case study for demonstration purposes. At each iteration step, the splitting approach discussed in Sec. 2.2 is used to derive the narrower bounds for the value of the constraint function $g_I(\boldsymbol{X})$ within each subregion. For demonstration purposes, the stopping criterion is set as the smallest region width in all dimensions is less than 10^{-5}. The stopping criterion is satisfied after ten iteration steps, and a total of 1,283,872 classified regions are output as sets of intervals. Among these 1,283,872 classified regions, there are 432,405 feasible regions, 429,280 infeasible regions, and 422,187 indeterminate regions. Notably, when the iterative process is halted, the area of the 422,187 indeterminate regions is less than 0.1% of the area of the whole design space. The 1,283,872 sets of intervals for the classified regions are provided in the public repository.[3]

All the 1,283,872 two-dimensional feasible, infeasible, and indeterminate regions are shown in Fig. 3(*a*). To demonstrate the planar visualization approach presented in Secs. 3 and 4, these classified regions are also projected into one figure with two horizontal axes, as shown in Fig. 3(*b*). Practitioners can observe the ranges for the two variables within which the constraint of the problem could be satisfied from the green blocks in the figure (i.e., $x_1 \in [-3.195, -0.098]$ and $[3.088, 6.185]$, $x_2 \in [-4.614, -1.517]$ and $[1.669, 4.766]$ in Fig. 3(*b*)). Notably, in this two-dimensional case study, each feasible region in Fig. 3(*a*) corresponds to two ranges within the corresponding green blocks in the two horizontal axes that are connected by an arrow, such as the feasible region 1 shown in Fig. 3(*b*). The two sets of intervals

[1] https://github.com/CMU-Integrated-Design-Innovation-Group/PITSA

[2] See Note 1.

[3] See Note 1.

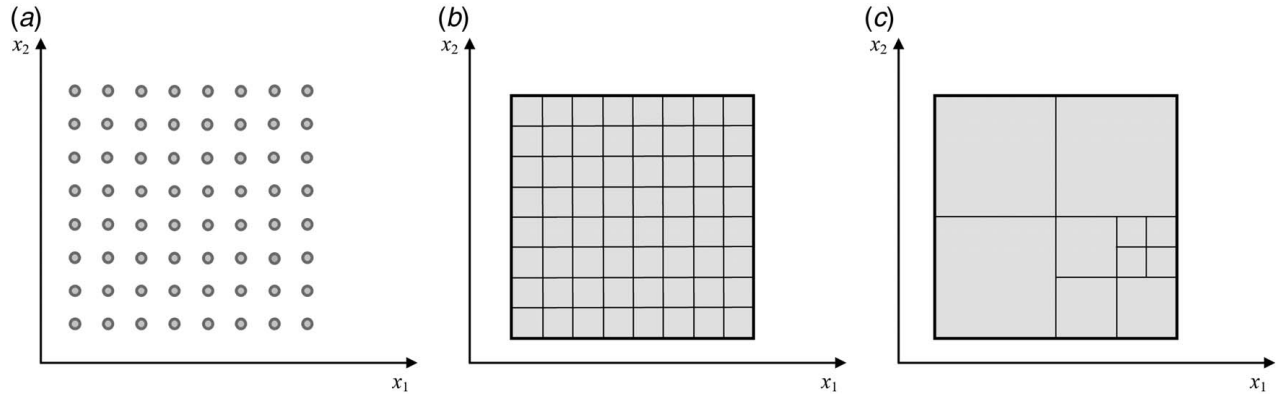

Fig. 4 Three methods to search the design space for continuous constraint satisfaction problems: (*a*) sampling-based exhaustive search, (*b*) region-based exhaustive search, and (*c*) region-based iterative search

Table 1 Computational times of three GPU-based methods for the bird function case study

Device	Sampling-based exhaustive search method	Region-based exhaustive search method	Parallel region classification method
Laptop with one GPU	~82,000 s	~127,000,000 s	1944 s
Server with one GPU	~46,600 s	~9,880,000 s	138 s

corresponding to the feasible region 1 can be found in the public repository.[4]

The effectiveness and efficiency of the GPU-based parallel region classification method could be compared with that of two GPU-based exhaustive search methods using the same computing resources. In this section, the performance of the GPU-based parallel region classification method is not compared with that of extant CPU-based methods since a fair comparison cannot be made by using different computing resources (i.e., CPU versus GPU in these two case studies) and a GPU-based method is usually significantly faster than its CPU-based counterparts. As illustrated in Fig. 4(*a*), the sampling-based exhaustive search method covers the whole design space with sample points, and the distance between two adjacent sample points is 10^{-5}. The value of the constraint function is calculated at these sample points in parallel on a GPU to check whether each sample point is a solution point. Notably, the sampling-based exhaustive search method is not able to determine whether the region enclosed by several sample points is a feasible region. In contrast, the region-based exhaustive search method, as illustrated in Fig. 4(*b*), partitions the design space as two-dimensional square areas with an edge length of 10^{-5}. These regions are evaluated and classified as feasible, infeasible, and indeterminate regions using interval arithmetic in parallel on a GPU. Importantly, the GPU-based parallel region classification method introduced in Sec. 3 is a region-based iterative search method, as illustrated in Fig. 4(*c*). The GPU-based parallel region classification method iteratively partitions the design space into many subregions. These subregions are evaluated and classified as feasible, infeasible, and indeterminate regions in parallel on a GPU at each iteration step, and the iteration process is halted when the smallest region width in all dimensions is less than 10^{-5}. Using these three GPU-based methods to solve the continuous constraint satisfaction problem defined by Eqs. (15)–(17), the results indicate that the GPU-based parallel region classification method derives the same feasible regions as the region-based exhaustive search method. In addition, using the same computing resource (a typical laptop with one GPU and a typical server with one GPU, respectively), the GPU-based parallel region classification method consumes significantly less computational time than both the sampling-based exhaustive search method and the region-based exhaustive search method. Specifically, as shown in Table 1, when the same server is employed as the computing resource, the GPU-based parallel region classification method consumes four orders less computational time than the region-based exhaustive search method but derives the same feasible regions for the problem; when compared to the sampling-based exhaustive search method, the GPU-based parallel region classification method only consumes 0.30% computational time but can derive feasible regions for the problem rather than only check whether the vertices of each square area are solution points. The laptop and server configurations are provided in Appendix B.

5.2 The Case Study of Welded Beam Design. The welded beam design is a benchmark engineering design problem [42–44], where a rectangular beam is designed to be welded onto a primary structure, as illustrated in Fig. 5. The original problem is adapted as a continuous constraint satisfaction problem including four design variables, $\boldsymbol{X} = \{x_1, x_2, x_3, x_4\}$, where x_1 is the weld height, x_2 is the length of the beam that is welded onto the primary structure, x_3 is the height of the beam, and x_4 is the thickness of the beam.

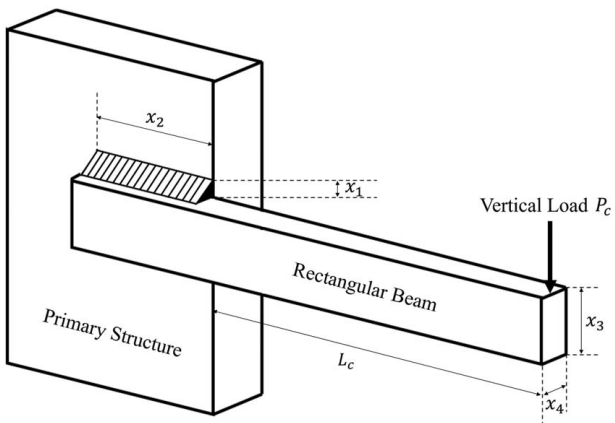

Fig. 5 Welded beam design problem

[4]See Note 1.

The allowable ranges of these four variables, $\boldsymbol{D} = \{d_1, d_2, d_3, d_4\}$, are defined in the unit of inch as:

$$\{d_1 : x_1 \in [0.001, 1]\} \tag{18}$$

$$\{d_2 : x_2 \in [0.001, 8]\} \tag{19}$$

$$\{d_3 : x_3 \in [5, 30]\} \tag{20}$$

$$\{d_4 : x_4 \in [0.001, 1]\} \tag{21}$$

Six constants involved in the welded beam design problem are provided in Table 2. Eight design constraints, $\boldsymbol{C} = \{c_1, c_2, \ldots, c_8\}$, are defined by

(1) The weld shear stress is less than or equal to 13,600 psi.

$$\{c_1 : g_1(\boldsymbol{X}) = \tau(\boldsymbol{X}) - 13{,}600 \le 0\} \tag{22}$$

$$\tau(\boldsymbol{X}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} \tag{23}$$

where τ' is the primary stress acting over the weld throat area defined by Eq. (24), and τ'' is the secondary torsional stress defined by Eqs. (25)–(28).

$$\tau' = \frac{P_c}{\sqrt{2}x_1x_2} \tag{24}$$

$$\tau'' = \frac{MR}{J} \tag{25}$$

$$M = P_c\left(L_c + \frac{x_2}{2}\right) \tag{26}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \tag{27}$$

$$J = 2\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right] \tag{28}$$

(2) The normal stress in the beam is less than or equal to 30,000 psi.

$$\{c_2 : g_2(\boldsymbol{X}) = \sigma(\boldsymbol{X}) - 30{,}000 \le 0\}, \tag{29}$$

$$\sigma(\boldsymbol{X}) = \frac{6P_cL_c}{x_4x_3^2} \tag{30}$$

(3) The weld height cannot exceed beam thickness.

$$\{c_3 : g_3(\boldsymbol{X}) = x_1 - x_4 \le 0\} \tag{31}$$

(4) The overall material cost cannot exceed \$5.

$$\{c_4 : g_4(\boldsymbol{X}) = S_1x_1^2x_2 + S_2x_3x_4(L_c + x_2) - 5 \le 0\} \tag{32}$$

(5) The weld height should be larger than or equal to 0.125 in..

$$\{c_5 : g_5(\boldsymbol{X}) = 0.125 - x_1 \le 0\} \tag{33}$$

Table 2 Six constants involved in the welded beam design problem

	Definition	Value
P_c	The vertical load applied on the beam	6000 pounds
L_c	Distance from the load to the point of support	14 in.
E_c	Young's modulus of the beam material	30×10^6 psi
G_c	Shear modulus of the beam material	12×10^6 psi
S_1	Weld material cost per unit volume	0.10471 dollars/in.^{3}
S_2	Beam material cost per unit volume	0.04811 dollars/in.^{3}

(6) The maximum allowable beam-end deflection is 0.25 in..

$$\{c_6 : g_6(\boldsymbol{X}) = \delta(\boldsymbol{X}) - 0.25 \le 0\} \tag{34}$$

$$\delta(\boldsymbol{X}) = \frac{4P_cL_c^3}{E_cx_3^3x_4} \tag{35}$$

(7) The 6000 lb vertical load does not cause the beam to buckle.

$$\{c_7 : g_7(\boldsymbol{X}) = 6000 - B(\boldsymbol{X}) \le 0\} \tag{36}$$

where the beam buckling load, $B(\boldsymbol{X})$, is derived by

$$B(\boldsymbol{X}) = \frac{4.013\sqrt{E_cG_c\frac{x_3^2x_4^6}{36}}}{L_c^2}\left(1 - \frac{x_3}{2L_c}\sqrt{\frac{E_c}{4G_c}}\right) \tag{37}$$

(8) The overall fabrication cost cannot exceed \$3.5.

$$\{c_8 : g_8(\boldsymbol{X}) = F(\boldsymbol{X}) - 3.5 \le 0\} \tag{38}$$

$$F(\boldsymbol{X}) = (1 + S_1)x_1^2x_2 + S_2x_3x_4(L_c + x_2) \tag{39}$$

To derive the feasible regions of the continuous constraint satisfaction problem defined by Eqs. (18)–(39), the GPU-based parallel region classification method presented in Secs. 3 and 4 is employed. The allowable ranges of the four variables given by Eqs. (18)–(21) define a four-dimensional design space for the problem. The whole design space is first evaluated using interval arithmetic. The code lists of the eight constraint functions, $g_j(\boldsymbol{X})$, are generated and provided in the public repository.[5] The sequence of interval arithmetic operations in each code list yields the outwardly rounded lower and upper bounds for the value of the constraint function $g_j(\boldsymbol{X})$ within the design space, $[\mathrm{DLB}_j, \mathrm{DUB}_j]$, where $j \in \{1, 2, \ldots, 8\}$, and the whole design space is classified as an indeterminate region accordingly. The design space is then iteratively partitioned into many subregions, and these subregions are evaluated and classified as feasible regions, infeasible regions, and indeterminate regions in parallel on the GPU of a server at each iteration step, where the server configuration is provided in Appendix B. The partition strategy defined by Eqs. (13) and (14) in Sec. 3 is employed in this case study for demonstration purposes. At each iteration step, the splitting approach discussed in Sec. 2.2 is used to derive the narrower bounds for the value of the constraint function $g_j(\boldsymbol{X})$ within each subregion. For demonstration purposes, the stopping criterion is set as the smallest region width in all dimensions is less than 0.05 in.. The stopping criterion is satisfied after seven iteration steps, and a total of 192,065,215 classified regions are output as sets of intervals. Among these 192,065,215 classified regions, there are 43,072,368 feasible regions, 59,961,905 infeasible regions, and 89,030,942 indeterminate regions. Notably, when the iterative process is halted, the four-dimensional volume of the 89,030,942 indeterminate regions is less than 0.1% of the volume of the whole design space. The 43,072,368 sets of intervals for the feasible regions are provided in the public repository.[6]

All the 192,065,215 four-dimensional feasible, infeasible, and indeterminate regions are also projected into one figure using the planar visualization approach presented in Secs. 3 and 4, as shown in Fig. 6. Practitioners can observe the ranges for the four variables within which all eight constraints could be satisfied from the green blocks in the figure (i.e., $x_1 \in [0.126, 0.736]$, $x_2 \in [1.238, 8]$, $x_3 \in [5, 20]$, and $x_4 \in [0.220, 0.752]$ in Fig. 6). In addition, once a specific range for a variable within a green block is specified by practitioners (e.g., $x_1 \in [0.251, 0.301]$ in Fig. 6), the ranges of other variables that lead to feasible regions in the design space are connected by arrows, and practitioners can generate or evaluate welded beam design concepts accordingly. The eight

[5]See Note 1.
[6]See Note 1.

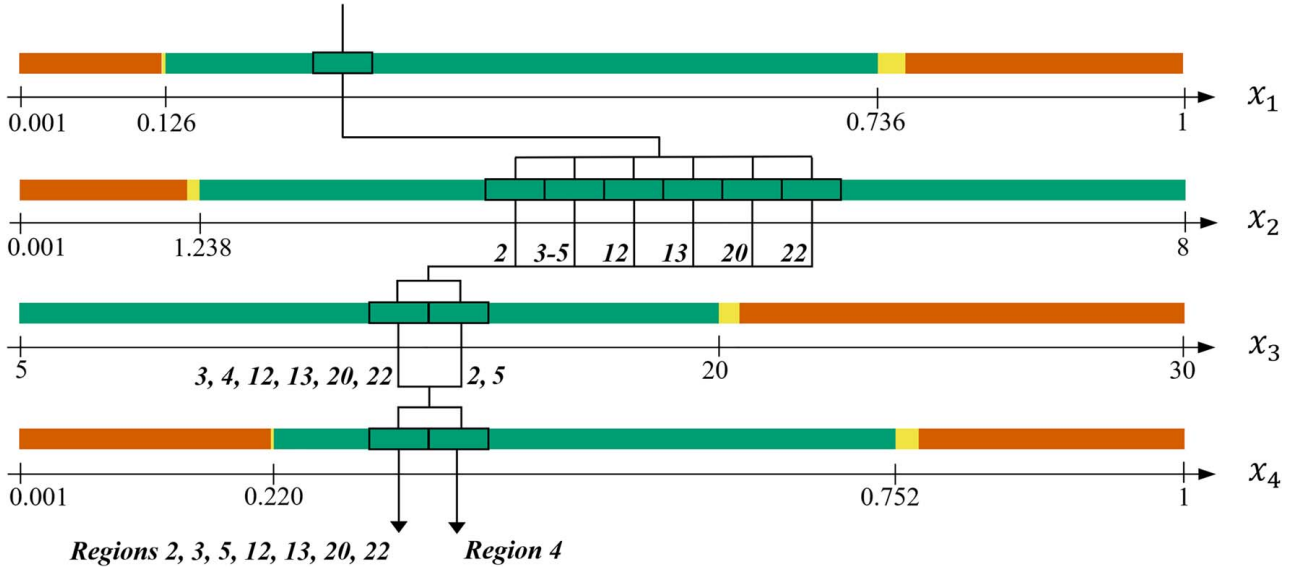

Fig. 6 Planar visualization for the result of the welded beam design case study

Table 3 Computational times of three GPU-based methods for the welded beam design case study

Device	Sampling-based exhaustive search method	Region-based exhaustive search method	Parallel region classification method
Laptop with one GPU	5032 s	~2,768,000 s	3883 s
Server with one GPU	2439 s	~446,000 s	431 s

sets of intervals corresponding to the feasible regions 2, 3–5, 12, 13, 20, and 22 in Fig. 6 can be found in the public repository.[7]

The effectiveness and efficiency of the GPU-based parallel region classification method could be compared with that of two GPU-based exhaustive search methods using the same computing resources. As illustrated in Fig. 4(*a*), the sampling-based exhaustive search method covers the whole design space with sample points, and the distance between two adjacent sample points is 0.05 in.. The values of the eight constraint functions are calculated at these sample points in parallel on a GPU to check whether each sample point is a solution point. Notably, the sampling-based exhaustive search method is not able to determine whether the region enclosed by several sample points is a feasible region. In contrast, the region-based exhaustive search method, as illustrated in Fig. 4(*b*), partitions the design space as four-dimensional hypercubes with an edge length of 0.05 in.. The regions enclosed by these hypercubes are evaluated and classified as feasible, infeasible, and indeterminate regions using interval arithmetic in parallel on a GPU. Importantly, the GPU-based parallel region classification method introduced in Sec. 3 is a region-based iterative search method, as illustrated in Fig. 4(*c*). The GPU-based parallel region classification method iteratively partitions the design space into many subregions. These subregions are evaluated and classified as feasible, infeasible, and indeterminate regions in parallel on a GPU at each iteration step, and the iteration process is halted when the smallest region width in all dimensions is less than 0.05 in.. Using these three GPU-based methods to solve the welded beam design problem defined by Eqs. (18)–(39), the results indicate that the GPU-based parallel region classification method derives the same feasible regions as the region-based exhaustive search method. In addition, using the same computing resource (a typical laptop with one GPU and a typical server with one GPU, respectively), the GPU-based parallel region classification method consumes significantly less computational time than both the sampling-based exhaustive search method and the region-based exhaustive search method. Specifically, as shown in Table 3, when the same server is employed as the computing resource, the GPU-based parallel region classification method consumes three orders less computational time than the region-based exhaustive search method but derives the same feasible regions for the problem; when compared to the sampling-based exhaustive search method, the GPU-based parallel region classification method only consumes 18% computational time but can derive feasible regions for the problem rather than only check whether the vertices of each four-dimensional hypercube are solution points. The laptop and server configurations are provided in Appendix B.

6 Conclusions and Future Work

A GPU-based parallel region classification method is introduced to solve continuous constraint satisfaction problems. The novelty of this method is that the method iteratively partitions the design space into many subregions based on the compute capability of the practitioner's GPU; at each iteration step, the method evaluates and classifies many subregions in parallel on the GPU. A planar visualization approach that projects all classified regions in the design space into one figure is also proposed for continuous constraint satisfaction problems. The application of the GPU-based parallel region classification method and the planar visualization approach is demonstrated through two case studies about the bird function and the welded beam design.

Using the GPU-based parallel region classification method, practitioners can overcome the computational bottleneck of extant CPU-based methods and derive the feasible regions as sets of intervals for continuous constraint satisfaction problems in a reasonable computational time. All these feasible regions are also projected into one figure by using the planar visualization approach, and the figure facilitates practitioners' design concept generation and evaluation.

This work has limitations that offer opportunities for future research. As stated in Sec. 3, the GPU-based parallel region classification method introduced in this paper applies to continuous constraint satisfaction problems with a finite set of inequality

[7]See Note 1.

constraints that have closed-form expressions, which covers many real-world engineering design problems (e.g., the welded beam design problem in Sec. 5.2). Future work could extend the method to continuous constraint satisfaction problems with both equality and inequality constraints, where each constraint may not have a closed-form expression (e.g., differential equations could be involved in one or multiple constraints). In addition, the partition strategy defined by Eqs. (13) and (14) is employed in the two case studies that involve two and four design variables (i.e., $n = 2$ and $n = 4$), respectively. Notably, the partition strategy defined by Eqs. (13) and (14) can lead to a significant number of subregions that need to be evaluated and classified at each iteration step for high-dimensional continuous constraint satisfaction problems (e.g., $n > 20$). In future research, the partition strategy could be modified based on the compute capability of the practitioner's GPU for high-dimensional problems. Using the modified partition strategy in future research, the computational costs of the GPU-based parallel region classification method in solving continuous constraint satisfaction problems with various dimensionality could be compared. Moreover, since the method proposed in this paper is based on interval arithmetic, the method is able to solve the continuous constraint satisfaction problems where one or multiple design variables are unbounded (e.g., $x \in [0, +\infty]$). Future research could introduce new partition strategies for the method to solve unbounded problems.

Acknowledgment

This material is partially supported by the Air Force Office of Scientific Research through grant FA9550-18-0088. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The authors attest that all data for this study are included in the paper. The code lists, the sets of intervals for the classified regions, and the Python source code to implement the GPU-based parallel region classification method for the two case studies in Sec. 5 are freely available for noncommercial research through the GitHub repository.[8]

Appendix A: Illustrative Figures for the Planar Visualization Approach

The projections from n-dimensional infeasible, indeterminate, and feasible regions to red, yellow, and green blocks for planar visualization are illustrated in Figs. 7–9, respectively.

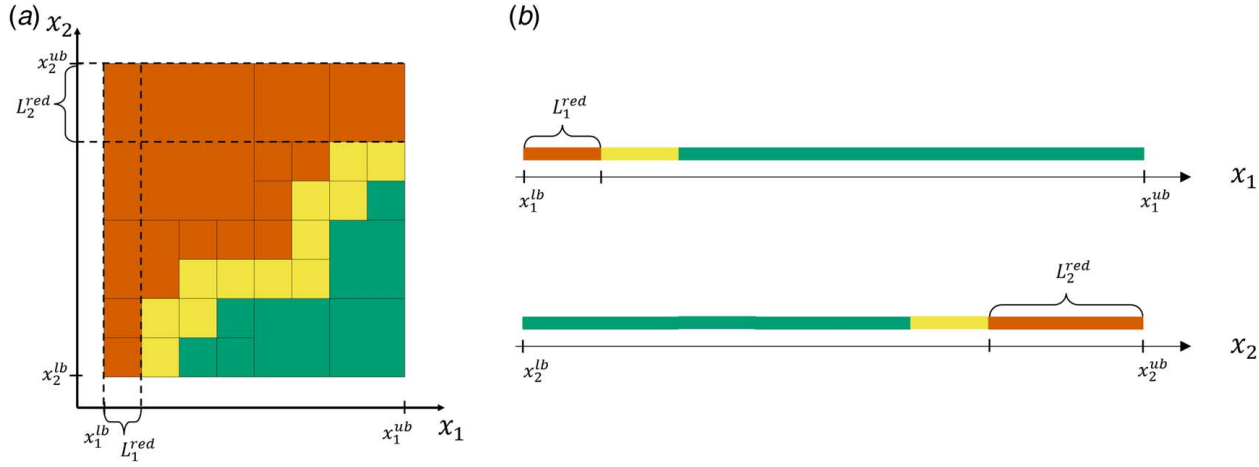

Fig. 7 Projection from infeasible regions to red blocks for planar visualization: (*a*) infeasible regions in the design space, (*b*) red blocks in the planar visualization figure

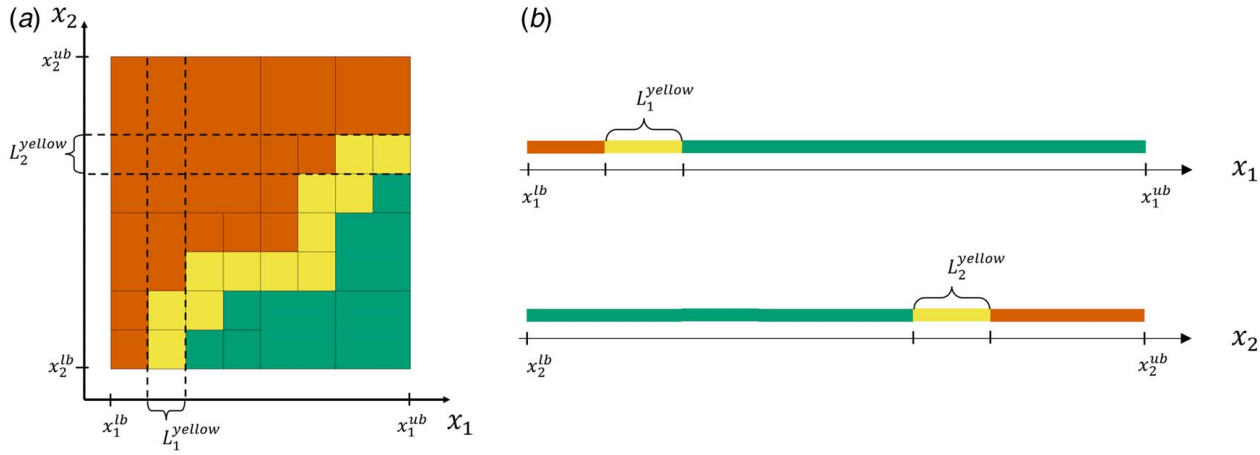

Fig. 8 Projection from indeterminate regions to yellow blocks for planar visualization: (*a*) indeterminate regions in the design space, (*b*) yellow blocks in the planar visualization figure

[8]See Note 1.

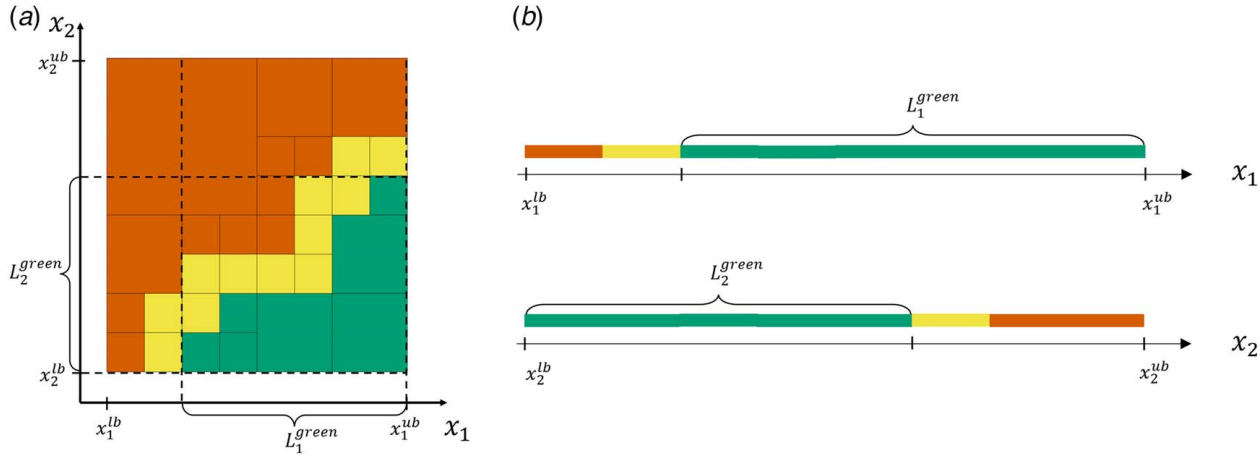

Fig. 9 Projection from feasible regions to green blocks for planar visualization: (*a*) feasible regions in the design space, (*b*) green blocks in the planar visualization figure

Appendix B: Laptop and Server Configurations

The configurations of the laptop and the server employed in the two case studies appear in Table 4.

Table 4 Laptop and server configurations

Device	CPU	RAM	GPU
Laptop	Intel Core i7-8750H	16 GB	NVIDIA GeForce GTX 1050 Ti
Server	AMD EPYC 7502	128 GB	NVIDIA Quadro RTX A6000

References

[1] Arróyave, R., Gibbons, S., Galvan, E., and Malak, R., 2016, "The Inverse Phase Stability Problem as a Constraint Satisfaction Problem: Application to Materials Design," JOM, **68**(5), pp. 1385–1395.

[2] Swaney, R. E., and Grossmann, I. E., 1985, "An Index for Operational Flexibility in Chemical Process Design. Part I: Formulation and Theory," AIChE J., **31**(4), pp. 621–630.

[3] Malan, S., Milanese, M., and Taragna, M., 1997, "Robust Analysis and Design of Control Systems Using Interval Arithmetic," Automatica, **33**(7), pp. 1363–1372.

[4] Yvars, P.-A., 2008, "Using Constraint Satisfaction for Designing Mechanical Systems," Int. J. Interact. Des. Manuf., **2**(3), pp. 161–167.

[5] Galvan, E., Malak, R. J., Gibbons, S., and Arroyave, R., 2017, "A Constraint Satisfaction Algorithm for the Generalized Inverse Phase Stability Problem," ASME J. Mech. Des., **139**(1), p. 011401.

[6] Larson, B. J., and Mattson, C. A., 2012, "Design Space Exploration for Quantifying a System Model's Feasible Domain," ASME J. Mech. Des., **134**(4), p. 041010.

[7] Simpson, T. W., Poplinski, J., Koch, P. N., and Allen, J. K., 2001, "Metamodels for Computer-Based Engineering Design: Survey and Recommendations," Eng. Comput., **17**(2), pp. 129–150.

[8] Wang, G. G., and Shan, S., 2007, "Review of Metamodeling Techniques in Support of Engineering Design Optimization," ASME J. Mech. Des., **129**(4), pp. 370–380.

[9] 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.

[10] Gelle, E., Faltings, B. V., Clément, D. E., and Smith, I. F., 2000, "Constraint Satisfaction Methods for Applications in Engineering," Eng. Comput., **16**(2), pp. 81–95.

[11] Lottaz, C., Clément, D. E., Faltings, B. V., and Smith, I. F., 1999, "Constraint-Based Support for Collaboration in Design and Construction," J. Comput. Civ. Eng., **13**(1), pp. 23–35.

[12] Granvilliers, L., and Benhamou, F., 2006, "Algorithm 852: Realpaver: An Interval Solver Using Constraint Satisfaction Techniques," ACM Trans. Math. Softw., **32**(1), pp. 138–156.

[13] Vu, X. H., 2005, "Rigorous Solution Techniques for Numerical Constraint Satisfaction Problems," Ph.D. thesis, Department of Computer Science, EPFL, Lausanne, Switzerland.

[14] Ratschan, S., 2006, "Efficient Solving of Quantified Inequality Constraints Over the Real Numbers," ACM Trans. Comput. Log., **7**(4), pp. 723–748.

[15] Hu, J., Aminzadeh, M., and Wang, Y., 2014, "Searching Feasible Design Space by Solving Quantified Constraint Satisfaction Problems," ASME J. Mech. Des., **136**(3), p. 031002.

[16] Sam-Haroud, D., and Faltings, B., 1996, "Consistency Techniques for Continuous Constraints," Constraints, **1**(1), pp. 85–118.

[17] Vu, X. H., Sam-Haroud, D., and Silaghi, M.-C., 2002, "Numerical Constraint Satisfaction Problems With Non-Isolated Solutions," *Global Optimization and Constraint Satisfaction. COCOS 2002. Lecture Notes in Computer Science*, Vol. 2861, C Bliek, C Jermann, and A Neumaier, eds., Springer, Berlin, Heidelberg, Germany.

[18] Goldsztejn, A., and Granvilliers, L., 2010, "A New Framework for Sharp and Efficient Resolution of NCSP With Manifolds of Solutions," Constraints, **15**(2), pp. 190–212.

[19] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C., 2008, "GPU Computing," Proc. IEEE, **96**(5), pp. 879–899.

[20] Apt, K., 2003, *Principles of Constraint Programming*, Cambridge University Press, Cambridge, UK.

[21] Tsang, E., 1993, *Foundations of Constraint Satisfaction*, Academic Press, London.

[22] Rossi, F., Van Beek, P., and Walsh, T., 2006, *Handbook of Constraint Programming*, Elsevier, Amsterdam, The Netherlands.

[23] Kumar, V., 1992, "Algorithms for Constraint-Satisfaction Problems: A Survey," AI Mag., **13**(1), pp. 32–32.

[24] Brailsford, S. C., Potts, C. N., and Smith, B. M., 1999, "Constraint Satisfaction Problems: Algorithms and Applications," Eur. J. Oper. Res., **119**(3), pp. 557–581.

[25] Chaloner, K., and Verdinelli, I., 1995, "Bayesian Experimental Design: A Review," Stat. Sci., **10**(3), pp. 273–304.

[26] Morris, M. D., and Mitchell, T. J., 1995, "Exploratory Designs for Computational Experiments," J. Stat. Plan. Inference, **43**(3), pp. 381–402.

[27] Pronzato, L., and Müller, W. G., 2012, "Design of Computer Experiments: Space Filling and Beyond," Stat. Comput., **22**(3), pp. 681–701.

[28] Hansen, E., and Walster, G. W., 2004, *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York.

[29] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J., 1999, "Revising Hull and Box Consistency," Logic Programming: Proceedings of the 1999 International Conference on Logic Programming, Las Cruces, NM, Nov. 29–Dec. 4, MIT Press, pp. 230–244.

[30] Sam, J., 1995, "Constraint Consistency Techniques for Continuous Domains," Ph.D. thesis, Department of Computer Science, EPFL, Lausanne, Switzerland.

[31] Han, H., Chang, S., and Kim, H., 2017, "A Systematic Approach to Identifying a Set of Feasible Designs," Proceedings of the 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Cleveland, OH, Aug. 6–9, DETC2017-68003, p. V02BT03A040.

[32] Moore, R. E., Kearfott, R. B., and Cloud, M. J., 2009, *Introduction to Interval Analysis*, SIAM, Philadelphia, PA.

[33] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E., 2001, *Applied Interval Analysis*, Springer-Verlag, London, UK.

[34] IEEE, 2015, *1788-2015 Standard for Interval Arithmetic*, IEEE, New York.

[35] Moore, R. E., 1962, "Interval Arithmetic and Automatic Error Analysis in Digital Computing," Ph.D. thesis, Department of Mathematics, Stanford University, Stanford, CA.

[36] IEEE, 2019, *754-2019 Standard for Floating-Point Arithmetic*, IEEE, New York.

[37] Kulisch, U. W., and Miranker, W. L., 1981, *Computer Arithmetic in Theory and Practice*, Academic Press, New York.

[38] Ratschek, H., and Rokne, J., 1984, *Computer Methods for the Range of Functions*, Halsted Press, New York.

[39] Rokne, J. G., 1986, "Low Complexity k-Dimensional Centered Forms," Computing, **37**(3), pp. 247–253.

[40] Neumaier, A., 1990, *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, UK.

[41] Jamil, M., and Yang, X., 2013, "A Literature Survey of Benchmark Functions for Global Optimisation Problems," Int. J. Math. Model. Numer. Optim., **4**(2), pp. 150–194.

[42] Ragsdell, K., and Phillips, D., 1976, "Optimal Design of a Class of Welded Structures Using Geometric Programming," ASME J. Eng. Ind., **98**(3), pp. 1021–1025.

[43] Rao, S. S., 2009, *Engineering Optimization: Theory and Practice*, John Wiley & Sons, Hoboken, NJ.

[44] Coello, C. A. C., 2000, "Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems," Comput. Ind., **41**(2), pp. 113–127.