

Srikanth Devanathan
School of Mechanical Engineering
Purdue University
dsrikanth@purdue.edu

Karthik Ramani
School of Mechanical Engineering
School of Electrical and Computer
Engineering (by Courtesy)
Purdue University
ramani@purdue.edu

Creating Polytope Representations of Design Spaces for Visual Exploration Using Consistency Techniques

Understanding the limits of a design is an important aspect of the design process. When mathematical models are constructed to describe a design concept, the limits are typically expressed as constraints involving the variables of that concept. The set of values for the design variables that do not violate constraints constitute the design space of that concept. In this work we transform a parametric design problem into a geometry problem thereby enabling computational geometry algorithms to support design exploration. A polytope-based representation is presented to geometrically approximate the design space. The design space is represented as a finite set of (at most) 3-dimensional (possibly non-convex) polytopes, i.e., points, intervals, polygons and polyhedra. The algorithm for constructing the design space is developed by interpreting constraint-consistency algorithms as computational geometric operations and consequently extending (3,2)-consistency algorithm for polytope representations. A simple example of a fingernail-clipper design is used to illustrate the approach.

1 Introduction

Many engineering design problems can be described parametrically using equations and inequalities that model, among other things, the physical behavior, feasibility and acceptability of that design. In general, three distinct subsets of the product parameters can be identified: design variables $\bar{X} = \{x_1, x_2, \dots, x_n\}$, performance parameters $\bar{P} = \{p_1, p_2, \dots, p_m\}$, and noise variables [1]. We can then define the design space, performance space, noise space, and concept space as follows (from Otto and Wood [1]):

- The *design space*, $D \subset \mathbb{R}^n$ is the set of considered possible alternative configurations (possibly empty), described using design variables, over which we have direct control. By definition, all points within the design space satisfy all constraints imposed on the design.
- The corresponding set of values of the performance parameters is called the *performance space*, $P \subset \mathbb{R}^m$ of the concept. Formal models ($M: D \rightarrow P$) map each design point $\bar{d} \in D$ to $M(\bar{d}) \in P$.
- The *noise space*, $N \subset \mathbb{R}^l$, is the set of possible configurations described using noise variables required to

evaluate any point in D , which we do not have direct choice over.

The design space can be empty, finite, or infinite. An empty design space signifies an infeasible or unacceptable design. Loosely speaking, the equality constraints in the model map a point in the design space onto a point in the performance space. Inequality constraints, including the bounds on the design variables, form the boundary of the design space (see Figure 1). Any point outside this design space is, by definition, infeasible. When a new system concept is composed of two or more different sub-system concepts (shortened to sub-concepts), more constraints are introduced to describe the compatibility among the sub-concepts. These constraints connect parameters of the sub-concepts as well as parameters that describe overall concept. To obtain a feasible design of the system, these constraints as well as those within each sub-concept have to be solved simultaneously.

Designers explore several ‘what-if’ scenarios to understand the design space and evaluate tradeoffs within the design. Design exploration involves changing the original problem and evaluating the effect of this change on the feasible design and performance spaces. Assigning values to certain design variables, modifying existing constraints (relaxing/tightening), adding new constraints, or

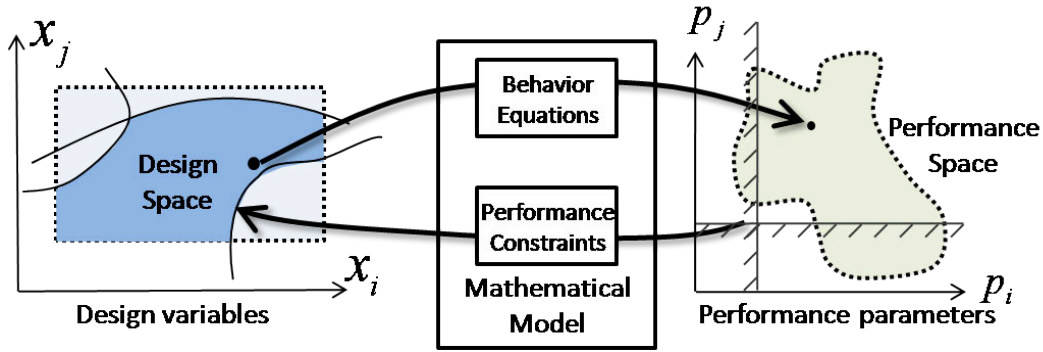


Figure 1. Formal (mathematical) model maps a point in the space of design variables to a point in the

removing existing constraints are some of the tasks performed by the user during exploration.

The fundamental issues in such design exploration are (1) the need for a compact representation of the design space and (2) efficient methods for creating such descriptions. Except for simple design problems, obtaining an exact representation of the design space is not tractable. However, if approximations are sufficient then computational methods can be developed to address these issues. When a design concept is described mathematically, the design problem can be posed as a constraint satisfaction problem and all the solutions to the constraint satisfaction problem constitute the design space of the original design problem.

In this paper, we use polytopes to represent design spaces; we then develop a consistency method to construct the approximation using computational geometric approach.

2 Related Work

Exploration of the design space is typically performed by sampling the mathematical model of the design at a finite set of design points and testing the points for feasibility. Design of experiments (DOE) and methods such as genetic algorithms have been used extensively for such sampling (Figure 2a). These are sometimes called “point-based design” methods. An extensive review of sampling strategies is available in [2]. These sample points are then filtered during the exploration process by modifying the constraints on the original design. Due to the computational cost of obtaining each feasible point, such point-based methods are either confined to a small region within the design space or provide a sparse sampling of the entire design space. Moreover, these samples do not lie on the boundary of the design space. Nevertheless, these points are then visualized using parallel coordinates, glyph plots and histograms among others [3].

When continuous design spaces are to be represented, constraint-programming techniques have been applied to obtain those design sets. Earliest among such methods is presented by Ward [4] using interval arithmetic to

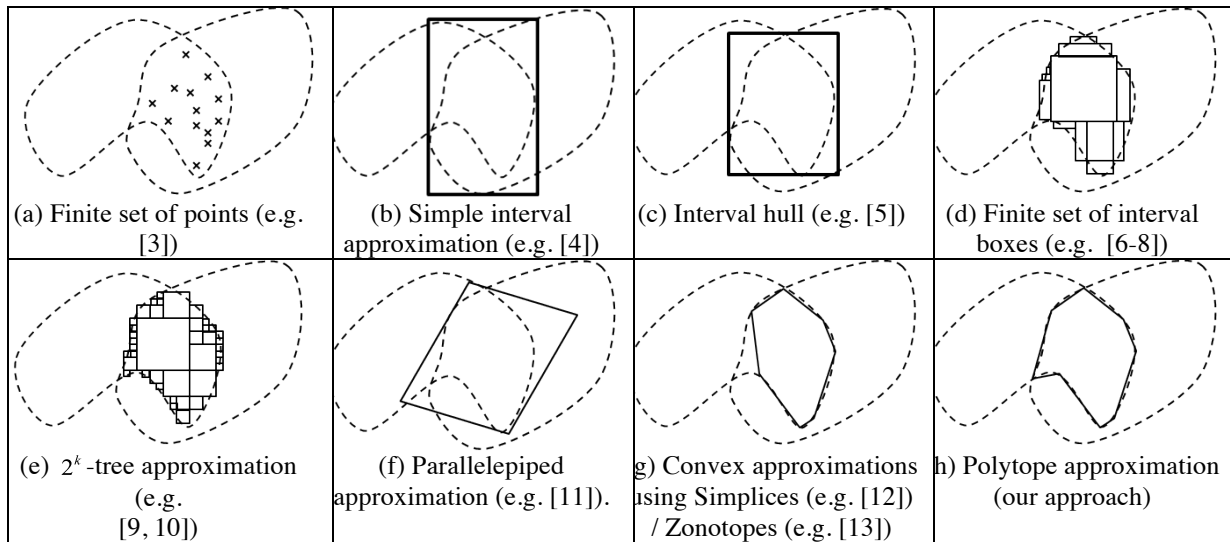


Figure 2. Different representations for solution and design spaces.

constraint interval representation of design solutions (Figure 2b). Extending simple intervals to interval hulls [5] (Figure 2c) and interval boxes allow approximation of solution sets using *orthogonal polyhedra* such as hulls of boxes and 2^k -trees. Yannou et al. [6] use RealPaver [7], an interval-based constraint solver, to generate a set of boxes that span the design space (Figure 2d). The algorithm continuously restricts the domain of the variables until a tight bounding box is reached. The algorithm then selects a variable (using round-robin policy) and partitions the current box in to n boxes along that variable axis. At each box, interval arithmetic is used to determine whether the entire box is feasible, infeasible or unknown. Unknown boxes are processed recursively until the required precision is obtained. Consistency methods such as hull-, and box-consistency are used to reduce the domain and obtain the solution space of the entire c-CSP. Fünzig et al. [8] use Linear Programming (LP) reductions on Bernstein polytopes instead of interval techniques to reduce the range of constraints and the domains of variables in order to compute the solution space of a finite set of non-linear constraints.

Sam-Haroud and Faltings [9] and subsequent work by Lottaz [10] use 2^k -trees (Figure 2e) to represent solution spaces of a continuous constraint satisfaction problem (c-CSP). The motivation for using 2^k -trees is cited as lack of algorithms using polyhedral data structures [14]. Since 2^k -trees are prohibitively expensive for higher dimensions, the n -ary¹ algebraic constraints are first transformed into a set of ternary constraints by introducing auxiliary variables. The solution space for each of these ternary constraints are constructed and stored as 2^k -trees. They use a pre-defined maximum tree depth for each problem. To construct the initial 2^k -tree ($k = 3$ as all constraints are now ternary) for each constraint, it is necessary to evaluate the feasibility of a constraint within the 3-block. Sam-Haroud and Faltings [9] use the gridding method described in [15] where each face of the block is partitioned into a grid and the constraint is evaluated at each grid point. The intersection of the constraint surface with the face of the block is determined and used for further partitioning. Lottaz [10] uses interval analysis to identify whether a block is feasible, infeasible or unknown. Only the blocks with “unknown” labeling are split recursively. Vu et al. [16] improve the efficiency of the algorithm by considering the complementary box and checking for infeasibility. Compression in representation of 2^k -trees is also provided by considering extreme vertices in orthogonal polyhedra.

Recently, Goldsztein and Granvilleirs [11] present a global constraint contraction technique using parallelepipeds (Figure 2f) rather than orthogonal polyhedra for constraints that are manifolds (i.e., the

constraints do not have ‘sharp’ corners or they do not self-intersect).

When the design space is convex (i.e., constraints are convex), Goyal [12] presents a “simplicial” approximation (a geometry where each face is a simplex) to represent the limits of feasible space (Figure 2g). Notions of inner and outer approximations are used to identify the operating envelope (design space) of a chemical process, i.e., the largest convex hull that can be inscribed within the feasible space. The inner approximation is any convex-hull that can be inscribed within the feasible space, and the outer approximation is the polytope formed by the tangent planes at the vertices of the inner approximation. The inner approximation is improved iteratively by comparing with the outer approximation. Zonotopes, a special case of convex polytopes constructed as the Minkowski sum of finite line segments, have also been used to obtain a convex bounding for the solution space of a constraint set [13].

Yan and Sawada [17] present a design exploration tool based on symbolic methods, namely Gröbner bases and quantifier elimination (QE). Although the method presented does not compute the entire solution space, it does provide lower dimensional basis polynomials that can be plotted and the solution space can be identified. Since QE and Gröbner bases are applicable only to polynomial constraints, transformations have also been suggested to generate polynomial approximations. Still, the computational complexity of QE and Gröbner bases are doubly exponential in number of variables, i.e. $O(e^e)$, and therefore can only be used for small problems.

In this work, we describe a geometric procedure for creating polytope approximations of the design spaces. It is conceivable that interval techniques are used to restrict the domain sufficiently prior to the construction of the polytope approximation; however, we assume that the domain is sufficiently narrowed before constructing an approximation of the design space using our method.

3 Creating Polytope Representation using Constraint Satisfaction Method

3.1. Constraint satisfaction problems

Constraint satisfaction problems (CSPs) are a class of problems designed to address systems that are primarily concerned with constraints. CSPs are modeled as constraint networks. A constraint network consists of a finite set of *variables* $X = \{x_1, \dots, x_n\}$ with respective *domains* $D = \{D_1, \dots, D_n\}$ and a set of *constraints* $C = \{C_1, \dots, C_r\}$ [18]. Constraints restrict the values that can be taken by the variables. These values belong to the domain of that variable. A *numeric* CSP is a special CSP where the domain for each variable is a subset of reals \mathbb{R} . Constraints can be specified explicitly as enumeration of valid combinations or in general as mathematical expressions.

¹ “n-ary” is the generalization of unary (1-ary), binary (2-ary), ternary (3-ary) and so on. “Arity” is defined in Section 3.

These constraint expressions can be equalities or inequalities, and can involve any number of variables.

The *arity* of a constraint is the number of variables involved in that constraint. For example, a unary constraint involves only one variable, binary constraint involves two variables and a ternary constraint involves three variables.

The objective of solving CSPs is to find a valid *instantiation* of the variables that does not violate any constraint. Solving a CSP, in general, is NP-Hard [18]; however, in certain special cases or when approximations are tolerated these methods become tractable.

The solution techniques for CSPs fall into two broad categories - Search and Filtering. The focus of search is on obtaining a single feasible (and possibly optimal) instantiation. Examples of search techniques include backtracking, branch and bound, and non-linear programming. Filtering or consistency involves identifying (to a certain extent, depending upon the actual algorithm) whether a CSP is feasible and to possibly obtain the set of all possible solutions, i.e., the *solution set*. Arc-consistency, path-consistency, box-consistency, and the (3,2)-relational consistency used here are examples of local consistency techniques for filtering. Arc- and path- consistency techniques are used for discrete problems where the domain is finite. Consistency methods guarantee that if they result in a null assignment (i.e., no solution) then the given problem does not have a solution. However, if no such conclusion is obtained then these methods do not guarantee that there is a solution to the given problem. For a detailed study of constraint processing techniques and their applications, the reader is referred to [18] and [19].

Under strong assumptions, global consistency techniques overcome such limitations. When the solution is globally consistent, then the variables of the problem can be assigned values from the solution set in a backtrack-free manner [18]. The (3,2)-relational consistency, for example, can guarantee global consistency when the solution spaces for the constraints are directionally convex. This condition is explained in Section 3.3.2.

3.2. Using polytopes to represent design space

As mentioned earlier, we assume that the design problem is represented using design variables, performance parameters, their domain and constraints between them. To construct the design space, we first construct the solution space for each constraint separately and then prune (filter) these solution spaces using a consistency technique to obtain the reduced labels for the constraints. Since the filtering algorithm guarantees that any point outside a label is infeasible, the resultant set of labels approximates the design space of the product concept.

A constraint of arity- k defines a set of points, called the solution space $L \subset \mathbb{R}^k$, where this constraint is satisfied. If the constraint is an equality constraint, then these points lie on a hyper-surface in k -dimensional space. Similarly, an

inequality constraint gives rise to regions (without regard to its connectivity) in k -dimensional space. Additionally, if the variables are bounded, i.e., their domain is prescribed using lower and upper bounds, then these surfaces and regions are bounded as well. L is also called the *label* corresponding to the k variables.

Since a design space is a region in n -dimensions, geometric data-structures can be used to represent such spaces. Although, the memory requirements and time complexity for representing and manipulating n -dimensional geometric objects increases exponentially with n [20], a n -dimensional region can be reduced to a collection of $n'(>n)$ three-dimensional geometric objects. This is because any algebraic expression involving k variables can be rewritten as a finite set of ternary expressions with an equivalent solution space [9]. As a simple example, consider the following expression of arity 4:

$$x_1 x_2 x_3 + x_2 x_4 - x_3^2 \leq 0 \quad (1)$$

This constraint can be re-written as four ternary constraints with the addition of two new variables $\{x_5, x_6\}$ as

$$\begin{aligned} x_5 &= x_1 x_2 \\ x_6 &= x_2 x_4 \\ x_5 x_3 + x_6 - x_3^2 &\leq 0 \end{aligned} \quad (2)$$

It is clear that any solution of eq. (1) is also a solution of the constraint set in eq. (2) and *vice versa*. In general, several intermediate variables may be required depending upon the coupling among the k variables in the original constraints. Lottaz et al. [21] present a method for introducing a minimal number of intermediate variables during the ternarization process. The consequence of restricting the maximum arity to three is that consistency needs to be maintained among additional 3-dimensional labelings. Consistency is maintained by iteratively pruning infeasible regions from the feasible space of individual constraints using a consistency algorithm until no further reduction is possible. The solution space of the entire CSP is the set of consistent labelings of each constraint.

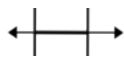

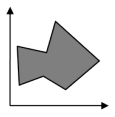
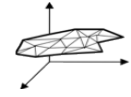
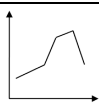
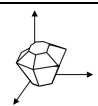
In this work, we use the term “polytope” to include: 2D polygonal region, 2D & 3D line segments, 3D polyhedral surface and 3D polyhedron as shown in Table 1.

3.3. Constructing the polytope approximation

The overall procedure for constructing the polytope approximation is:

1. Formulate the design problem as a numeric CSP
2. Transform all constraints into ternary constraints
3. Create the solution space for each constraint.
4. Use consistency to prune the solution space to obtain the design space.

Table 1. Examples of polytopes

Type of Constraint	Type of solution space		Type of Constraint	Type of solution space	
$f(x) \leq 0$	1D Interval		$f(x, y) = 0$	2D Polygon	
$f(x, y) \leq 0$	2D Polygonal region		$f(x, y, z) = 0$	3D Polyhedral surface (possibly with boundary)	
$f(x, y) = 0$	2D Polyline		$f(x, y, z) \leq 0$	3D Polyhedron	

3.3.1 Creating the initial solution space for each constraint

Given a (ternary) constraint C_{ijk} involving variables $\{x_i, x_j, x_k\}$ and an initial feasible starting point P_{start} , we construct the solution space around this point using the procedure shown in Figure 3. The basic idea here is to start with a small facet polyhedron (sphere in line 1) around the start point and iteratively “push” its vertices along the direction of the outward normal till the constraint boundary is reached (lines 4 and 5). Whenever a facet size exceeds a pre-set limit (λ), the facet is subdivided introducing new vertices (see Figure 4). A similar procedure is used for other polytopes. Figure 5 illustrates the filling process.

Procedure generate-label

Input: Variables (i, j, k) , Function check – feasibility(\langle point \rangle),

Feasible starting point P_{start} , Max. facet length λ

Output: Polytope label L

- 1 $L_{ijk} \leftarrow$ faceted sphere with center P_{start} and radius λ ;
- 2 **foreach** point $P \in L_{ijk}$ that is not on constraint boundary **do**
- 3 $N_P \leftarrow$ surface normal to L_{ijk} at P ;
- 4 $s_{max} \leftarrow$ Max s s.t. $\{Max\|(P + sN_P) - P_t\| = \lambda$ and
check – feasibility($P + sN_P$) is feasible};
- 5 $P \leftarrow P + sN_P$;
- 6 **foreach** facet $f \in L_{ijk}$ s.t. $P \in f$ **do**
- 7 **if** Max – chord – length(f) $> \lambda$ **then**
- 8 Subdivide f ;
- 9 **return** L_{ijk}

Figure 3. Procedure to construct a polyhedral label involving variables

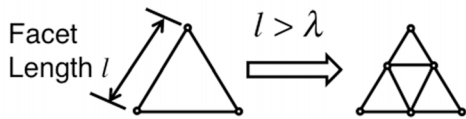


Figure 4. Subdividing a facet into smaller facets

The function check-feasibility is used in line 4 to test the feasibility of a point $P = (\alpha_i, \alpha_j, \alpha_k)$ by evaluating the constraint C_{ijk} . A control parameter λ , represents the maximum length allowed for a facet.

Complexity of procedure generate-label

Without loss of generality, we can assume that the design variables are suitably scaled such that the domain is a box of unit length. We estimate the computational complexity by assuming that the 3D space consists of points that are exactly λ apart – there are $n_p = O(d^3)$ such

points, where $d = \left\lceil \frac{1}{\lambda} \right\rceil$. In the worst case, all these points

lie on the boundary of the label (or solution space). Therefore the maximum number of facets that can exist is $O(d^3)$. Now, if each of these points are obtained by stepping the original seed point located at one of the corners of the bounding box, then each point is stepped $O(d)$ times. The constraints are therefore checked $O(d^4)$ times from which the complexity of the filling algorithm is $O(d^4)$.

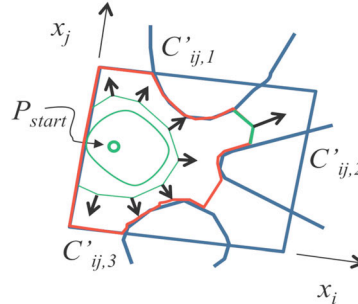


Figure 5. Illustration of the “filling” procedure to construct solution space around a starting point.

3.3.2 Pruning the solution space to obtain design space

Once the individual solution spaces are constructed, the (3,2)-consistency algorithm [9] is used to remove infeasible points from the solution space of infeasible points. The function *init* initializes the queue Q with permutations of 5 variables and L with the initial labels of all triplets of variables.

The main idea in (3,2)-relational consistency is to use two variables (u,v) involved in constraints C_{iuv}, C_{juv} and C_{kuv} to reduce the labeling of constraint C_{ijk} . The constraints involving combinations of $\{x_i, x_j, x_k, x_u, x_v\}$ are generated in the function *related-5tuples*.

Figure 6 shows the (3,2)-relational consistency algorithm for reducing the labels of the constraints. The queue Q initially contains the list of all related 5-tuples. The labelings associated with variables belonging to the 5-tuples are pruned iteratively in the function (3,2)-*revise* (Figure 7). Here, a polyhedron in dimensions $\{x_i, x_j, x_k\}$ is constructed by convolution of L_{iuv}, L_{juv} and L_{kuv} followed by projection on to $\{x_i, x_j, x_k\}$ space and finally intersecting this projection with the existing $\{x_i, x_j, x_k\}$ -label, i.e. L_{ijk} .

Convolution, also called external join, is the operation a space of higher dimension from those of lower dimensions using common dimensions. Figure 8 illustrates the convolution operation in three dimensions. More precisely, convolution is defined as:

$$L_{iju} \otimes_u L_{klu} = \{(\alpha_i, \alpha_j, \alpha_k, \alpha_u) \mid (\alpha_i, \alpha_j, \alpha_u) \in L_{iju} \ \& \ (\alpha_k, \alpha_i, \alpha_u) \in L_{klu}\} \quad (3)$$

$$L_{iuv} \otimes_{uv} L_{juv} = \{(\alpha_i, \alpha_j, \alpha_u, \alpha_v) \mid (\alpha_i, \alpha_u, \alpha_v) \in L_{iuv} \ \& \ (\alpha_j, \alpha_u, \alpha_v) \in L_{juv}\} \quad (4)$$

Algorithm 2: (3,2)-relational-consistency(C)

Input: Constraints C
Output: Labels L

- 1 $Q, L \leftarrow \text{init}(C)$;
- 2 **while** $Q \neq \emptyset$ **do**
- 3 $(i, j, j, u, v) \leftarrow \text{pop}(Q)$; /* remove 5-tuple from queue */
- 4 **if** (3,2)-*revise*(i, j, k, u, v) **then**
- 5 /* if the label is actually pruned */
- 5 $Q \leftarrow Q \cup \text{related-5tuples}(i, j, k)$; /* propagate this change to other related labels */
- 6 **return** L

Figure 6. (3,2)-consistency algorithm to prune the solution spaces

Function (3,2)-*revise*($\{C\}$)

Input: Constraints $\{C\}$
Output: Queue Q , Set of labels $\{L\}$

- 1 $Q \leftarrow \{(i, j, k, u, v) \mid i, j, k, u, v \in X, i \leq j \leq k, u \leq v\}$ $L_{ijk} \leftarrow L'_{ijk} \leftarrow L_{ijk} \cap \left(\prod_{ijk} L_{iuv} \otimes L_{juv} \otimes L_{kuv} \right)$; /* extended convolution operation */
- 2 **if** $L'_{ijk} = L_{ijk}$ **then**
- 3 **return** *FALSE*; /* the label has not changed */
- 4 **return** *TRUE*; /* the label has changed */

Figure 7. Function to revise the solution L_{ijk} using the solution spaces L_{iuv}, L_{juv} and L_{kuv} .

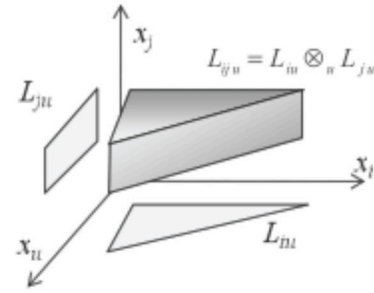


Figure 8. Convolution in three dimensions.

Step 1 of (3,2)-*revise* implicitly creates a temporary 5-dimensional geometric object in $\{x_i, x_j, x_k, x_u, x_v\}$ -space in order to compute $L_{ijk} \cap \left(\prod_{ijk} L_{iuv} \otimes_{uv} L_{juv} \otimes_{uv} L_{kuv} \right)$. This extended convolution operation (convolution followed by projection) reduces the original label L_{ijk} to L'_{ijk} . To propagate this change, other constraints that share a common variable are then added to the queue for revision by related-5tuples.

However, L'_{ijk} can also be constructed geometrically using the procedure *generate-label* by suitably modifying the *check-feasibility* function as follows: From eq. (4), it can be deduced that a point $P = (\alpha_i, \alpha_j, \alpha_k)$ belongs to L'_{ijk} if there is *at least one* value of $(x_u, x_v) = (\alpha_u, \alpha_v)$ such that the points $(\alpha_i, \alpha_u, \alpha_v), (\alpha_j, \alpha_u, \alpha_v)$ and $(\alpha_k, \alpha_u, \alpha_v)$ lie within L_{iuv}, L_{juv} and L_{kuv} respectively. Therefore, in order to check the feasibility of a point P , it is sufficient to check if such a (α_u, α_v) exists, i.e.,

$$\text{check-feasibility}(P) = \left(\prod_{uv} L_{iuv} \mid_{x_i = \alpha_i} \right) \cap \left(\prod_{uv} L_{juv} \mid_{x_j = \alpha_j} \right) \cap \left(\prod_{uv} L_{kuv} \mid_{x_k = \alpha_k} \right) \quad (5)$$

Figure 9 shows how $\prod_{uv} L_{uv} |_{x_i=\alpha_i}$ is obtained geometrically; similar sections can be obtained for variables x_j and x_k . The function generate-label returns TRUE if the intersection of the uv projections is not empty and FALSE otherwise.

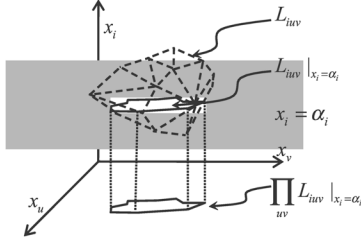
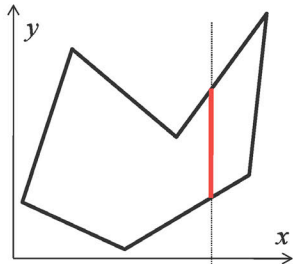


Figure 9. Geometrically, $\prod_{uv} L_{uv} |_{x_i=\alpha_i}$ is computed by obtaining an intersecting L_{uv} with the plane $x_i = \alpha_i$

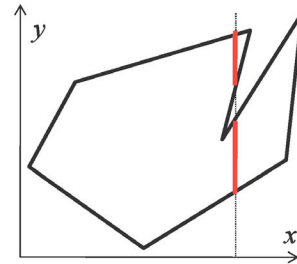
Convexity considerations for (3,2) – relational consistency

As mentioned in Section 3.1, the (3,2)-relational consistency algorithm used here, computes a globally consistent solution when the constraints are directionally convex. A set in \mathbb{R}^n is convex if its intersection with every straight line is connected or empty [22]. This condition is relaxed for directional-convexity, where only lines parallel to a given set of lines are considered [23]. A special case of directional convexity is called ortho-convexity [23] where the lines are parallel to the axes. Figure 10 illustrates this concept for planar shapes; (x,y) -, (y,z) - or (x,z) -convexity can be similarly defined for ternary spaces.

It is noted that, in order to choose the values for variable x_{k+1} having chosen values for variables x_1, x_2, \dots, x_k in a backtrack-free manner from the solution space computed by (3,2) consistency, each of the ternary solution spaces involving variables x_{k+1} and x_i (with $i \leq k$) should be (x_{k+1}, x_i) -convex [9]. (3,2)-relational consistency algorithm will fail to detect inconsistency in the solution space when the solution space does not satisfy the partial convexity condition.



a. Polygon is x -convex but not y -convex.



b. Polygon is neither x -, nor y -convex.

Figure 10. Example of directional convexity in planar shapes.

Complexity of function (3,2) – revise

Since the same generate-label procedure is used to compute the extended convolution of three constraints, $O(d^3)$ is also upper bound for the number of times the convolution condition (eq. 5) is checked. However, since this condition is checked geometrically using plane-polyhedron intersection ($O(d^3)$ [20]) followed by polygon intersection ($O(d^2)$, [20]), the complexity of calculating the extended convolution is $O(d^7)$.

3.3.3 Checking feasibility of a design point

The design space approximation constructed can be used to check the feasibility of a design point. Assigning values to a set of design variables may specify the design point. Such assignments are considered as additional constraints that are added to the original problem. For example, if $x_i = \alpha_i$ is an assignment, then the constraint

C_{uv} and its corresponding label L_{uv} involving variables x_i, x_u and x_v is replaced by a label L_{uv} given by

$\prod_{uv} L_{uv} |_{x_i=\alpha_i}$ (Figure 9). Therefore, a 3D polytope is

reduced to a 2D polygon or a set of points; a 2D polygon is reduced to a line segment or a set of points.

This change is then propagated to all other labels using the (3,2)-consistency algorithm described in Section 3.3.2. The point is infeasible when a null label is obtained, and therefore does not belong in the design space.

4 Example: Fingernail Clipper Design

Consider the design of fingernail clipper from Otto and Wood [1]. The main customer requirements on the clipper are: (1) easy to use, (b) should be compact, and (c) should have a long life. Figure 11 shows one of the many concepts developed to satisfy these customer requirements. The user applies a force on the pads that cause the arm to pivot about the end. The blades on the other end of the clipper shear the nail. The requirements are translated into constraints on the finger force, material stress, and overall dimensions.

4.1 Formulate the design problem as a numeric CSP

The mathematical model of this design concept is developed by considering the shear stress (τ) needed to cut a fingernail. The finger-force (f) needed to actuate the clipper is expressed in terms of the pivot length (L_p), width of the blade (W), nail thickness (t), distance of the grip from the pivot (x) and height of blade (h_b) as:

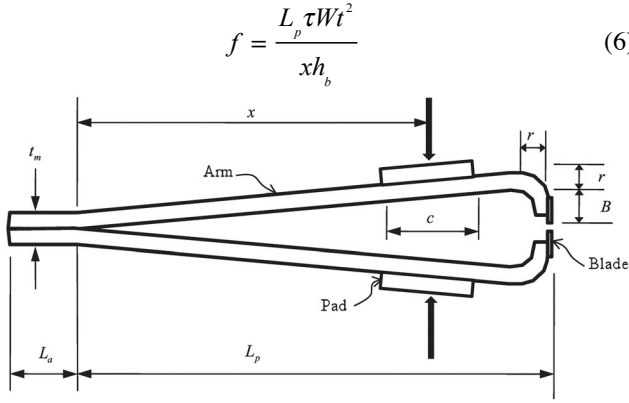


Figure 11. Schematic of fingernail clipper design (adapted from Otto and Wood [1]).

The maximum normal stress in the arm is given by:

$$\sigma = K_{cycles} \frac{6xf}{Wt_m^2} \left(1 - \frac{x}{L_p}\right) \quad (7)$$

Finally, the geometry of the concept gives rise to the following relations:

$$L_p = x + \frac{c}{2} + r \quad (8)$$

$$r = 2t_m \quad (9)$$

$$L_{overall} = L_a + L_p \quad (10)$$

The complete mathematical model of this design concept is developed considering the behavior and structure (geometry of the design) is given by

$$\begin{aligned} f &= \frac{L_p \tau W t^2}{x h_b} & L_p &= x + \frac{c}{2} + r \\ r &= 2t_m & L_{overall} &= L_a + L_p \\ \sigma &= K_{cycles} \frac{6xf}{Wt_m^2} \left(1 - \frac{x}{L_p}\right) & H_{overall} &= 2r + 2B + d \\ & & B &= h_b + 3D_h \end{aligned} \quad (11)$$

The constraints on the design concept are

$$\begin{aligned} f &\leq f_{max} & 5D_h &\leq W \\ 5t_m &\leq L_a & \sigma &\leq \sigma_{max} \\ h_b + 1.1t &\leq d \end{aligned} \quad (12)$$

The five design variables in this problem are $\{t_m, D_h, h_b, x, d\}$. The domains for the design variables are

$$\begin{aligned} t_m &\in [0.01, 0.06]; D_h \in [0.01, 0.06]; h_b \in [0.05, 0.5]; \\ x &\in [0.05, 5]; d \in [0.1, 0.2] \end{aligned} \quad (13)$$

The value of the constants are

$$\begin{aligned} \tau &= 1000 \text{ psi}; t = 0.025 \text{ in}; c = 0.375 \text{ in}; \\ L_{overall} &= 2.5 \text{ in}; W_{overall} = 0.5 \text{ in}; H_{overall} = 0.688 \text{ in}; \\ \sigma_{max} &= 15000 \text{ psi}; K_{cycles} = 5.0 \end{aligned} \quad (14)$$

We eliminate unnecessary intermediate variables using eq. (11) to obtain the following set of constraints involving design variables and constants.

$$\begin{aligned} \left(x + \frac{c}{2} + 2t_m\right) \tau W t^2 & - f_{max} \leq 0 \\ \frac{x h_b}{5D_h - W} & \leq 0 \\ 4t_m + 2h_b + 6D_h + d - H & \leq 0 \\ L_a + x + \frac{c}{2} + 2t_m - L & \leq 0 \\ 5t_m - L_a & \leq 0 \\ K_{cycles} \frac{6\tau}{h_b} \left(\frac{c}{2} + 2t_m\right) - \sigma_{max} & \leq 0 \end{aligned} \quad (15)$$

Clearly, the design space is a region in \mathbb{R}^5 . For sake of clarity, we denote the design variables by

$X = \{x_i\}$ where $\{x_1 \equiv t_m, x_2 \equiv D_h, x_3 \equiv h_b, x_4 \equiv x, x_5 \equiv d\}$. To explore this 5-dimensional design space, we create a set of 3-dimensional projections and maintain consistency among these projections based on the geometry of the design space.

4.2 Transform all constraints into ternary constraints

We notice that the constraint $4x_1 + 2x_3 + 6x_2 + x_5 - H \leq 0$ involves 4 design variables. By introducing a new variable x_6 and a new constraint,

$$C_{256} : x_6 = 6x_2 + x_5 \quad (16)$$

we can rewrite the original problem as:

$$C_{134} : \frac{\tau W_{overall} t^2 \left(x_4 + \frac{c}{2} + 2x_1\right)}{x_4 x_3} - f_{max} \leq 0 \quad (17)$$

$$\text{i.e., } \tau W_{overall} t^2 \left(x_4 + \frac{c}{2} + 2x_1\right) - x_3 x_4 f_{max} \leq 0$$

$$C_{346} : 4x_4 + 2x_3 + x_6 - H_{overall} \leq 0 \quad (18)$$

$$C_2 : 5x_2 - W_{overall} \leq 0 \quad (19)$$

$$C_{14} : 7x_1 + x_4 + \frac{c}{2} - L \leq 0 \quad (20)$$

$$C_{13} = K_{cycles} \frac{6\tau}{x_3} \left(\frac{c}{2} + 2x_1\right) - \sigma_{max} \leq 0 \quad (21)$$

$$\text{i.e., } K_{cycles} 6\tau \left(\frac{c}{2} + 2x_1\right) - x_3 \sigma_{max} \leq 0$$

Now, all the constraints have *arity* of three or less, i.e., involve three or fewer variables each.

4.3. Create the solution space for each constraint

Next, we construct the solution space for each of these constraints (initial labelings) using the procedure given in

Section 3.3.1. These initial labels for constraints C_{134} , C_{256} and C_{346} are shown in Figures 12, 13 and 14 respectively. The lambda value was kept at 0.05 for all the solution spaces.

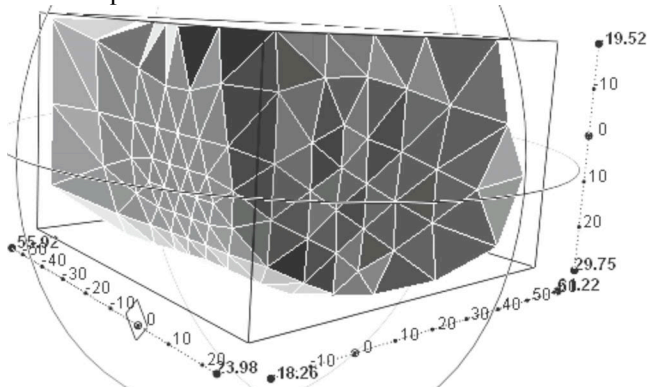


Figure 12. Initial label of ternary constraint

$$C_{134} \equiv \{t_m, h_b, x\}$$

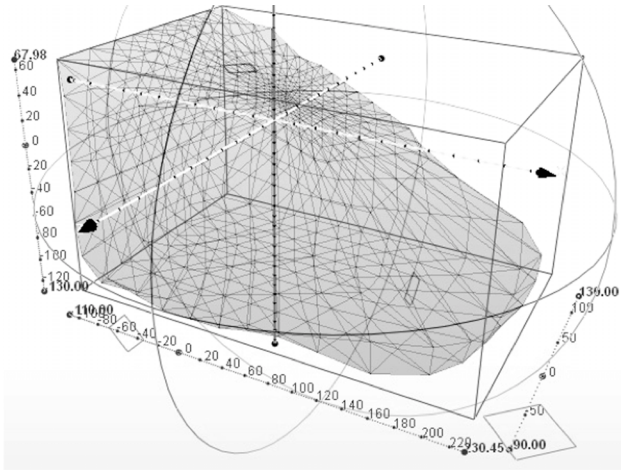


Figure 13. Initial label of ternary constraint

$$C_{346} \equiv \{h_b, x, x_6\}$$

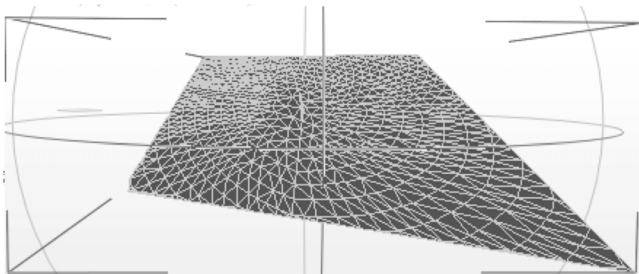


Figure 14. Initial label of ternary constraint

$$C_{256} \equiv \{D_h, d, x_6\}$$

4.4 Use consistency to prune the solution space to obtain the design space

Figures 15 and 16 show the labeling created after applying the consistency algorithm on the individual

solution spaces. These together represent the design space of the fingernail clipper concept.

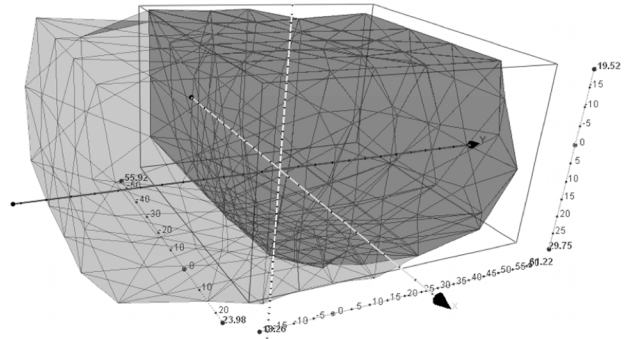


Figure 15. The initial (light) and final (dark) label of constraint $C_{134} \equiv \{t_m, h_b, x\}$

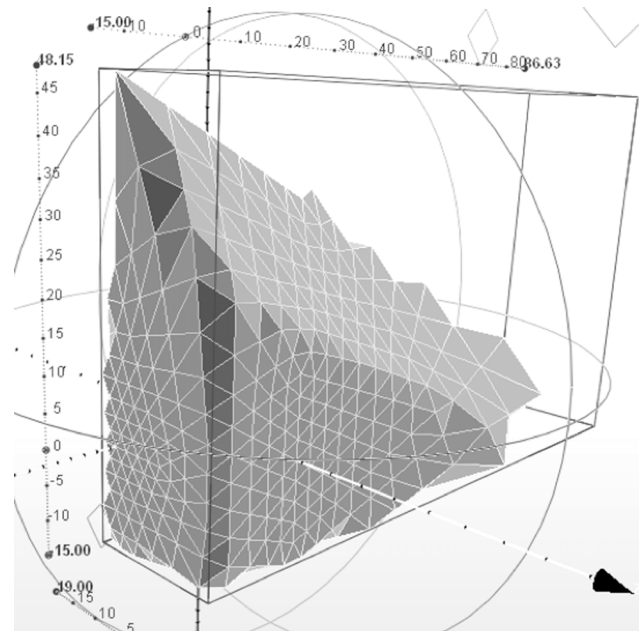


Figure 16. The reduced label of constraint

$$C_{346} \equiv \{h_b, x, x_6\} (f_{\max} = 3\text{lbs}).$$

4.5 Exploration of the design space by modifying f_{\max}

The labeling of the constraint C_{134} , is modified by changing the upper bound of f , i.e., f_{\max} . Whenever a particular constraint labeling is modified, the change is propagated to other labelings to maintain consistency. The resulting labelings involving (x_3, x_4, x_6) , i.e., L_{346} are shown in Figures 17 and 18. We note that when f_{\max} is reduced beyond (approximately) 1.8lbs, the labelings reduce to an empty set, i.e., 1.8lbs is a lower bound for the maximum force. This result agrees with that from [1] where the minimum value of f_{\max} is obtained at around 2 lbs.

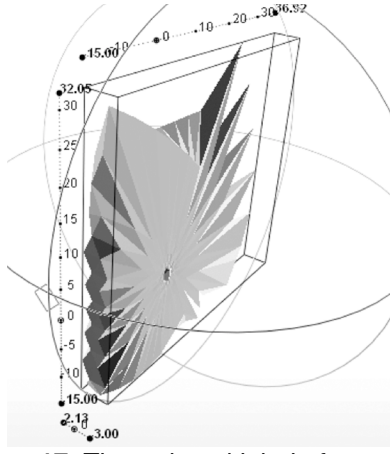


Figure 17. The reduced label of constraint

$$C_{346} \equiv \{h_b, x, x_6\} (f_{\max} = 2.5\text{lbs}, \lambda=0.05).$$

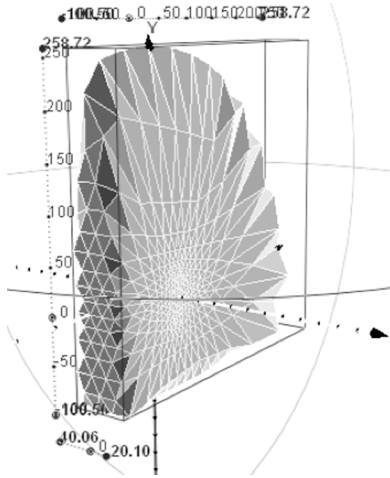


Figure 18. The reduced label of constraint

$$C_{346} \equiv \{h_b, x, x_6\} (f_{\max} = 2.1\text{lbs}, \lambda=0.025).$$

5 Validation of design space Approximation

The accuracy of any approximation technique used in multi-disciplinary optimization (such as Response Surfaces or Kriging) is assessed using the following error measures [25]:

1. Trust-region calculations
2. Hypothesis-testing methods such as ANOVA
3. Loss-function methods, and
4. Cross-validation methods.

The above measures cannot be applied to the design space approximation as it is a binary predictor: given a point in the space of design variables, the approximation predicts whether that point belongs to the design space or not. Counting the number of correct and incorrect predictions for a finite set of known samples and calculating metrics using those counts have been used to estimate the accuracy of binary predictors in the field of data-mining [26].

In order to assess the quality of the approximation, we generated a random set of points in the space of design

variables whose inclusion in the original design space is checked by evaluating the algebraic constraints. The prediction computed by the design space approximation was then be classified as one of:

1. True Positive (TP) – A point is correctly predicted as belonging to the design space
2. True Negative (TN) – A point is correctly predicted as not belonging to the design space
3. False Positive (FP) – An infeasible point is wrongly predicted as belonging to the design space.
4. False Negative (FN) – A point in the design space is incorrectly excluded.

Based on these counts, the percentage errors and the correlation coefficient for the binary samples were calculated (from [26]):

1. Percentage errors i.e., the number of true predictions for every hundred queries can be defined in three ways:
 - a. Total prediction percentage (PCT) measures the percentage of correct predictions to the total sample size.

$$PCT = 100 \frac{TP + TN}{TP + TN + FP + FN} \quad (22)$$

- b. Positive prediction percentage (PCP) measures the percentage of true positive predictions to the total number of feasible points. This number can be interpreted as the confidence value when a positive prediction is obtained.

$$PCP = 100 \frac{TP}{TP + FN} \quad (23)$$

- c. Negative prediction percentage (PCN) measures the percentage of true negative predictions to the total number of infeasible points. This number can be interpreted as the confidence value when a negative prediction is obtained.

$$PCN = 100 \frac{TN}{TN + FP} \quad (24)$$

2. The correlation coefficient (C) measures how much the predictions agree with the actual values. A correlation of -1 indicates total disagreement, +1 indicates total agreement and a zero correlation indicates a random prediction [26]. This number compares the prediction algorithm to a random assignment of positive and negative values to the sample points.

$$C = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \quad (25)$$

We also applied the same metrics to solution spaces obtained using a simple implementation of 2^k -tree representation ($k \leq 3$), convex approximations, and finite set of interval boxes [6] using Realpaver v 1.0 [7]. Only the “unknown” boxes whose edge length exceed λ were partitioned into smaller boxes. The convex approximation

was constructed using the procedure described in [12] and the maximum facet length was restricted to λ .

Table 2 summarizes the results of the study. It is evident that decreasing the λ value produces better approximations, irrespective of the representation used. Since the design space of the fingernail clipper is convex, both the convex and the polytope approximations are indistinguishable. Both the 2^k -tree and box representations have no false negatives but more false positives for the same resolution (λ -value). This is consistent with the intent of those representations, i.e., not to exclude any feasible solution.

6 Discussion

A key feature of the proposed consistency method is that it requires only the ability to check feasibility of a point with respect to the constraint, i.e., no secondary information like gradients are necessary. Additionally, unlike methods proposed by Sam-Haroud and Faltings [9] and Lottaz [10], the method presented here approximates equality relationships using surfaces as opposed to regions. Although the objective in [9, 10] is to obtain tighter bounds for the solution space, our objective here is to obtain a tighter approximation of the design space.

Although analytical constraints are used in the illustrative example, this method can be extended to other causal models such as Finite Element Analysis by using appropriate surrogate models such as response surfaces. The choice of meta-models is expected to determine the quality of approximation obtained because of couplings among variables similar to interval techniques [6].

During the course of the case study, limitations of the proposed technique were identified. These limitations deal with the quality of approximation generated. Trivially, an approximation can be improved by decreasing the values of the tuning parameter λ - at the cost of memory and speed. Firstly, during the construction of solution spaces, several attempts were needed before an acceptable value of λ could be chosen. This problem is not unique to the approach presented here; indeed, all approximation methods use sampling, and the sampling size determines the quality of the resulting approximation.

Second aspect, what we call ‘edge-effect’ wherein saw-tooth like surface is encountered around a limit state, results in a poor approximation by increasing both FP and FN. This effect is clearly visible in Figure 17, when the size of the labeling is comparable to λ . This was overcome in Figure 18 by using a smaller value of λ .

Third, although this was not observed for the case study, geometric cases can exist where large portions of the design space are ignored by the approximation method. Figure 19 illustrates one such case.

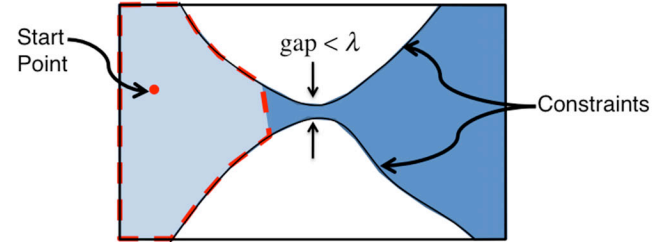


Figure 19. Erroneous approximations can be obtained for large values of λ .

Fourth, since the solution space is generated with a single seed point, only the connected region around this point is explored. If the solution space consists of multiple regions, then the technique presented in this article may incorrectly predict infeasibility.

7 Concluding Remarks

The key idea in this work is leveraging computational-geometric algorithms for constraint processing by using equivalent geometric operations for constraint processing. We have described, in this article, a geometric algorithm for constructing a polytope approximation of the design space. The approach involves transforming a parametric design problem into a geometry problem and thereby using computational geometry algorithms to support design exploration. Here, the parametric design problem is first transformed into a ternary-constraint satisfaction problem; then, the solution space of each of the constraints is created using a ‘‘filling’’ procedure. The initial solution spaces so created are subsequently pruned using a consistency technique. Future work for improving the technique will include developing an adaptive mesh generation strategy to

λ	Set of interval boxes [6,7]				2^k - tree [8]				Convex approximation [12]				Polytope			
	PCP	PCN	PCT	C	PCP	PCN	PCT	C	PCP	PCN	PCT	C	PCP	PCN	PCT	C
0.05	55.0	29.3	84.3	0.7117	52.0	34.4	86.4	0.7537	59.8	36.3	96.1	0.9207	59.3	35.9	95.2	0.8997
0.025	60.7	31.5	92.2	0.8428	57.6	37.1	94.7	0.8951	58.7	39.1	97.8	0.9552	59.5	38.5	98.0	0.9587
0.0125	58.6	38.5	94.5	0.8926	56.9	38.9	95.8	0.9168	58.6	39.2	97.9	0.9553	59.3	39.1	98.4	0.9632

reduce the number of facets needed to approximate the design space without compromising accuracy. An interactive visualization and exploration tool is also needed for enhancing the utility of the proposed technique in design. Another future direction is also to use the Geometric Processing Unit (GPU) to increase the speed of constraint operations for design space exploration.

8 Acknowledgements

The authors acknowledge the support of Discovery Park Center for Advanced Manufacturing (CAM) at Purdue University, for partially supporting the work presented. We also thank the anonymous reviewers for the insightful comments in improving the paper.

9 References

- [1] Otto, K., and Wood, K., 2001, *Product Design: Techniques in Reverse Engineering and New Product Development*, Prentice Hall, Upper Saddle River, NJ.
- [2] 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.
- [3] Stump, G. M., Yukish, M., Simpson, T. W., and O'Hara, J. J., 2004, "Trade Space Exploration of Satellite Datasets Using a Design by Shopping Paradigm," *Proc. IEEE Aerospace Conference*, 6, pp. 3885- 3895.
- [4] Ward, A. C., 1989, "A Theory of Quantitative Inference Applied to a Mechanical Design Compiler," Ph. D. thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Boston, MA.
- [5] Benhamou, F., 1995, "Interval Constraint Logic Programming," *Proc. Constraint Programming: Basics and Trends*, LNCS 910, pp. 1-21.
- [6] Yannou, B., Simpson, T. W., and Barton, R. R., 2003, "Towards a Conceptual Design Explorer Using Meta-Modeling Approaches and Constraint Programming," *ASME Design Engineering Technical Conferences, DETC2003/DAC-48766*, Chicago, IL.
- [7] Granvilliers, L., and Benhamou, F., 2006, "Algorithm 852: Realpaver: An Interval Solver Using Constraint Satisfaction Techniques," *ACM Transactions on Mathematical Software*, 32(1), pp. 138-156.
- [8] Fünfzig, F., Michelucci, D., Foufou, S., 2009, "Nonlinear Systems Solver in Floating-Point Arithmetic using LP Reduction," *Proc. 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, San Francisco, California, pp. 123-134.
- [9] Sam-Haroud, D., and Faltings, B., 1996, "Consistency Techniques for Continuous Constraints," *Constraints*, 1(1/2), pp. 85-118.
- [10] Lottaz, C., 2000, "Collaborative Design Using Solution Spaces," Ph. D. Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.
- [11] Goldsztein, A., and Granvilliers, L., 2008, "A New Framework for Sharp and Effective Resolution of Ncsp with Manifolds of Solutions," *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008*, Sydney, Australia, September 14-18, 2008.
- [12] Goyal, V., 2005, "Design and Synthesis of Flexible Module Based Systems," Ph. D. thesis, Department of Chemical Engineering, Rutgers, The State University of New-Jersey, New Brunswick, NJ.
- [13] Combastel, C., 2003, "A State Bounding Observer based on Zonotopes", *European Control Conference 2003*, Cambridge, England.
- [14] Jermann, C., Neumaier, A., and Sam, D., 2005, *Global Optimization and Constraint Satisfaction*, Springer-Verlag, Berlin/Heidelberg.
- [15] Mortenson, M. E., 2006, *Geometric Modeling*, Industrial Press.
- [16] Vu, X.-H., Sam-Haroud, D., and Silaghi, M.-C., 2003, "Numerical Constraint Satisfaction Problems with Non-Isolated Solutions " *Springer Berlin / Heidelberg, Valbonne-Sophia Antipolis, France, October 2-4, 2002*.
- [17] Yan, X.-T., and Sawada, H., 2006, "A Framework for Supporting Multidisciplinary Engineering Design Exploration and Life-Cycle Design Using Underconstrained Problem Solving," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(4), pp. 329-350.
- [18] Dechter, R., 2003, *Constraint Processing*, Morgan Kaufmann Publishers, San Francisco, CA.
- [19] Rossi, F., Van Beek, P., Walsh, T., 2006, *Handbook of Constraint Programming*, Elsevier.
- [20] Samet, H., 2006, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, San Mateo, CA.
- [21] Lottaz, C., Smith, I. F. C., Robert-Nicoud, Y., and Faltings, B. V., 2000, "Constraint-Based Support for Negotiation in Collaborative Design," *Artificial Intelligence in Engineering*, 14, pp. 261-280.
- [22] Agoston, M. A., 2005, *Computer Graphics & Geometric Modeling: Mathematics*, Springer, London, England.
- [23] Fink, E., and Wood, D., 1996, "Fundamentals of Restricted-Oriented Convexity," *Information Sciences*, 92, pp. 175-196.
- [24] www.cgal.org, *Computational Geometric Algorithms Library*, accessed February, 2009
- [25] Tenne, Y., 2008, "Metamodel Accuracy Assessment in Evolutionary Optimization," *Proc. of IEEE World Congress on Computational Intelligence-WCCI 2008*, pp.1505-1512.
- [26] Baldi, P., Brunak, S., Chauvin, Y., Anderson, C.A.F., Nielson, H., 2000, "Assessing the Accuracy of Prediction Algorithms for Classification: An Overview," *Bioinformatics Review*, 16(5), pp. 412-424.