Approximating solution spaces as a product of polygons

H. Harbrecht, D. Tröndle and M. Zimmermann

Departement Mathematik und Informatik
Fachbereich Mathematik
Universität Basel
CH-4051 Basel

Preprint No. 2019-13
December 2019

www.math.unibas.ch

APPROXIMATING SOLUTION SPACES AS A PRODUCT OF POLYGONS

HELMUT HARBRECHT, DENNIS TRÖNDLE, AND MARKUS ZIMMERMANN

Abstract. Solution spaces are regions of good designs in a potentially high-dimensional design space. Good designs satisfy by definition all requirements that are imposed on them as mathematical constraints. In previous work, the complete solution space was approximated by a hyper-rectangle, i.e., the Cartesian product of permissible intervals for design variables. These intervals serve as independent target regions for distributed and separated design work. For a better approximation, i.e., a larger resulting solution space, this article proposes to compute the Cartesian product of two-dimensional regions, so-called 2d-spaces, that are enclosed by polygons. 2d-spaces serve as target regions for pairs of variables and are independent of other 2d-spaces. A numerical algorithm for non-linear problems is presented that is based on iterative Monte-Carlo sampling.

1. Introduction

Some technical systems, such as vehicles or airplanes, are difficult to design because of complexity: many interacting components from different technical disciplines with uncertain properties are to be arranged and adjusted such that the overall system behavior satisfies overall system requirements and the system reaches its design goal [31].

Established methods such as *sensitivity analysis* [27] or *multidisciplinary design optimization* [19] address design complexity due to many design variables, however without uncertainty treatment. Uncertainty is considered in *robust design optimization* [1] or *reliability-based design optimization* [25]. These methods are, however, only applicable when a detailed uncertainty model is available and is equipped with appropriate data, e.g., on the density distribution of random variables.

Development procedure models, such as the so-called V-model, see [17], provide a framework for so-called top-down design without uncertainty model. Technical systems are decomposed into smaller and more manageable parts. These parts may be seen as sub-systems or sub-sub-systems, etc., and will be referred to as components. In order to direct the design work on components towards the overall design goals, component requirements are formulated. They have to be such that they are (1) sufficient for reaching the overall

system goal, (2) as little restrictive for component design as possible, and (3) independent of component interaction.

Requirements can be expressed quantitatively as so-called *solution spaces*, [32]. Solution spaces are sets of *good designs*, i.e., design points that satisfy all system level-requirements. Solution spaces are typically approximated as axis-parallel hyper-rectangles, called solution boxes. They are maximized in order to enclose uncertainty and provide design flexibility. The edges of a solution box represent permissible regions for each design variable. As long as each uncertain design variable assumes a value from within its associated permissible interval the overall system goal is reached. Design variables on components are considered to be *decoupled* in a particular sense: their interaction is not relevant anymore as long as they stay within their intervals.

Expressing decoupled component requirements with solution spaces enables separated component development in different teams. The interval widths provide room for design flexibility and to cope with uncertainty. Tedious coordination and iteration between teams may be thus avoided. Practical applications can be found for vehicle crash design in [9, 12], for the design of vibrating systems in [20, 23], for chassis components in [7, 30], for product family design in [6], and for control systems design in [21, 22].

There are several algorithms that compute solution spaces as high-dimensional axis-parallel solution boxes. A detailed analysis of the basic algorithm introduced in [32] can be found in [14]. In [10], linear support vector machines are utilized to find hyperplanes that represent the good design space. Then, hypercubes are computed that lie within this good design space. In [26], a combination of a cellular evolutionary strategy and interval arithmetic is applied to find a hyper-rectangle. However, this technique requires that the objective function is known explicitly, which is not the case in this article. An algorithm proposed in [2] uses fuzzy set theory and cluster analysis to find a hypercube. On the downside, it requires a so-called *membership function*, which cannot be given for a design process as presented in this article.

In some cases, solution boxes are too small for practical applications and larger solution spaces are desirable. One approach to alleviate this problem is to divide the design variables into so-called *early-* and *late-decision* variables and enlarge the solution space for early-decision variables by compensating with late-decision variables, for details see [29]. Another approach relies on so-called *2d-spaces* where only pairs of design variables are decoupled: their permissible regions would then be two-dimensional regions. An algorithm to compute convex and piece-wise linear 2d-spaces for linear problems was proposed in [8]. In [16], design variables were also coupled and the 2d-spaces were represented as rotated boxes (as opposed to axis-parallel boxes).

This paper aims at extending the approach based on 2d-spaces. An algorithm will be presented that computes a solution space as the product of arbitrary 2-dimensional polygon-shaped 2d-spaces. As *polytopes* are the equivalent of polygons in higher dimensions, the problem addressed in this paper is referred to as *polytope optimization*.

The rest of this article is structured as follows. The problem statement is introduced in Section 2. Section 3 gives the overview to manipulations of polygons, which are used later in the polytope optimization algorithm. The polytope optimization algorithm is then proposed in Section 4. In Section 5, numerical examples are provided. Finally, conclusions are drawn in Section 6.

2. Problem Statement

Properties of product components are measured by *design variables* $x_i \in \mathbb{R}$. The overall design is given by the possibly very high-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$. The performance of a design is evaluated by an *objective function* $f : \Omega_{\text{ds}} \to \mathbb{R}$, where $\Omega_{\text{ds}} \subset \mathbb{R}^d$ is the space of all admissible designs. The function f measures the performance of the design, e.g. related to passenger safety or vehicle dynamics. This leads to the optimization problem

$$f(\boldsymbol{x}) \to \min_{\boldsymbol{x} \in \Omega_{\text{ds}}} . \tag{2.1}$$

The evaluation of f may be quite expensive, since it describes a complex numerical simulation. Therefore, it is mandatory to keep the number of function evaluations small. Moreover, f is a black-box function. It might be noisy and no information about its gradient is available.

The box optimization algorithm replaces problem (2.1) and instead tries to find a hyperbox $\Omega_{\text{box}} = \prod_{k=1}^{d} [a_k, b_k]$ such that all designs $\boldsymbol{x} \in \Omega_{\text{box}}$ fulfil the relaxed optimality criterion $f(\boldsymbol{x}) \leq c$ for a given threshold value $c \in \mathbb{R}$. The corresponding optimization problem reads (compare [11, 16])

$$\begin{cases} \text{Maximize the volume } \mu\left(\Omega_{\text{box}}\right) \\ \text{over all axis-parallel boxes } \Omega_{\text{box}} \subset \Omega_{\text{ds}} \\ \text{subject to } f(\boldsymbol{x}) \leq c \text{ for all } \boldsymbol{x} \in \Omega_{\text{box}}. \end{cases} \tag{2.2}$$

In order to work with this problem, the following definitions are introduced (compare again [11, 16]):

Definition 2.1. A design $\boldsymbol{x}$ is called a *good design* or a *good design point* if $f(\boldsymbol{x}) \leq c$ and *bad design* or *bad design point* if $f(\boldsymbol{x}) > c$ for a critical threshold value $c \in \mathbb{R}$ which is given. Additionally, the set of all good designs is defined as the complete solution space

$$\Omega_c := \{\boldsymbol{x} \in \Omega_{\text{ds}} : f(\boldsymbol{x}) \leq c\} .$$

For example, if f is the Rosenbrock function,

$$f(\boldsymbol{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2,$$

then Ω_c is U–shaped for $c = 20$, see Figure 1. An axis-parallel box will settle somewhere in the lower curve of the U–shape, which ignores a large part of the good design space Ω_c (see left plot in Figure 1). By contrast, a solution space with the shape of a polygon with an arbitrary number of vertices is able to adjust to the U–shape, and approximates the complete solution space better (see right plot in Figure 1).

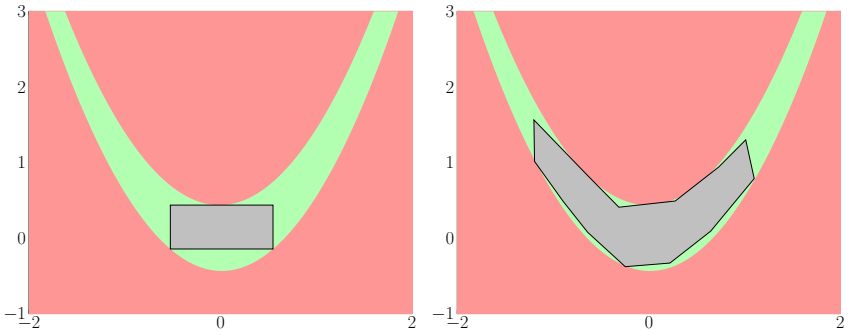

FIGURE 1. An axis-parallel box (left) and a polygon (right) inside the U-shape.

In order to allow for polytopes as solution spaces, the concept of *2d-spaces* is utilized. The idea of 2d-spaces is to couple pairs of design variables x_i and x_j. Then, instead of searching for two separate intervals $[a_i, b_i]$ and $[a_j, b_j]$ that satisfy the optimization problem (2.2), one can try to find a joint set $S_{i,j}$ in the associated 2d-space such that each pair $(x_i, x_j) \in S_{i,j}$ is part of a good design. Formally, a 2d-space can be defined as follows.

Definition 2.2. The *2d-space* $\Omega_{i,j}$ is defined as

$$\Omega_{i,j} := \left\{ \boldsymbol{y} \in \mathbb{R}^2 : \boldsymbol{y} = \pi_{i,j}(\boldsymbol{x}),\, \boldsymbol{x} \in \Omega_{\text{ds}} \right\},$$

where $\pi_{i,j}$ is the projection $(x_1, \ldots, x_d) \mapsto (x_i, x_j)$ with $1 \leq i, j \leq d$. That is, the 2d-space $\Omega_{i,j}$ is the projection of Ω_{ds} onto the dimensions i and j.

Note that have aready been introduced in [8]. In the polytope optimization algorithm, this concept of 2d-spaces is utilized and the joint set that is computed on each 2d-space is a polygon. Thus, the axis-parallel hyperbox Ω_{box} is replaced by a solution space Ω_{pol}, which is a product of one-dimensional intervals I_k and two-dimensional polygons $P_{i,j}$,

$$\Omega_{\text{pol}} := \prod_{k \in \mathcal{J}_{\text{int}}} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j}, \tag{2.3}$$

where, with $\mathcal{J}_P = \{1, \dots, d\} \setminus \mathcal{J}_{\text{int}}$, we have

$$\mathcal{J}_{\text{int}} = \big\{ i \in \{1, \dots, d\} : i \text{ is an unpaired dimension} \big\},$$
$$\mathcal{J}_{\text{pair}} = \big\{ (i,j) \in \mathcal{J}_P \times \mathcal{J}_P : \text{the dimensions } i \text{ and } j \text{ are coupled} \big\}.$$

The solution space Ω_{pol} is thus a specific high-dimensional polytope. As an example, Figure 2 illustrates a solution space Ω_{pol} in three dimensions. If Ω_{pol} is a product of polygons only, i.e.

$$\Omega_{\text{pol}} = \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j},$$

it is a *product prism*, which is the term for a polytope that is a product of polytopes with two or more dimensions (see [4]). Likewise, we define a polytope that can be written in the fashion of (2.3) as a *product polytope*. We define the space of *admissible* product polytopes as

$$\Omega_{\text{prod}} := \left\{ \Omega_{\text{pol}} \subset \Omega_{\text{ds}} \,\middle|\, \Omega_{\text{pol}} = \prod_{k \in \mathcal{J}_{\text{int}}} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j} \right\}.$$

Finally, problem (2.2) can be rewritten in the following way:

$$\text{(2.4)} \qquad \begin{cases} \text{Maximize the volume } \mu\left(\Omega_{\text{pol}}\right) \\ \text{over all polytopes } \Omega_{\text{pol}} \in \Omega_{\text{prod}} \\ \text{subject to } f(\boldsymbol{x}) \le c \text{ for all } \boldsymbol{x} \in \Omega_{\text{pol}}. \end{cases}$$

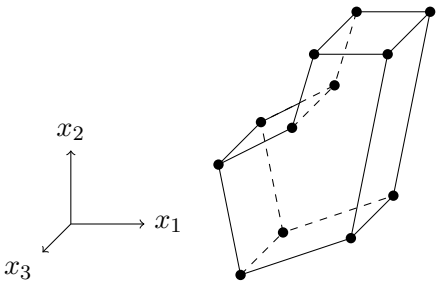

FIGURE 2. Visualization of a polytope $\Omega_{\text{pol}} = I_3 \times P_{1,2}$ in three dimensions, where $I_3 = [0, 1.5]$ and $P_{i,j} = \{(0,-1), (1.5,-0.5), (2,2), (1,2), (0.7,1), (-0.3,0.5)\}$.

As explained in [16], loss of flexibility by coupling pairs of design variables is accepted, because the volume of the polytope is expected to be much larger when compared to an axis-parallel hyperbox. We emphasize that the design problem at hand in general yields a natural choice for the design variables that should be coupled. For example, it is reasonable to couple design variables that one designer has full access to. Design variables are paired with at most one other design variable, and are thus represented on

at most one 2d-space. Design variables that have not to be paired with another variable are assigned to an interval I_k.

3. Manipulating Two-Dimensional Polygons

This section intends to give an overview of the basic manipulation steps of two-dimensional polygons. They are the the basic ingredients for the polytope optimization algorithm presented in Section 4. A polygon P has a fixed number N of vertices and is represented by the ordered sequence of vertices, that is

$$P = \left\{ \boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(N)} \right\}.$$

3.1. **Sample Points.** A design point inside the polygon P is obtained by constructing a bounding box around P and sampling uniformly distributed random points inside the bounding box until a point $\boldsymbol{x} \in \mathbb{R}^2$ is found that also lies within P. Determining whether a point lies within a polygon can be done via the winding number algorithm (compare [18]). After a fixed number of design points have been sampled, all design points $\boldsymbol{x}$ are evaluated with the objective function f and then marked as good and bad points, compare Figure 3 for an illustration.

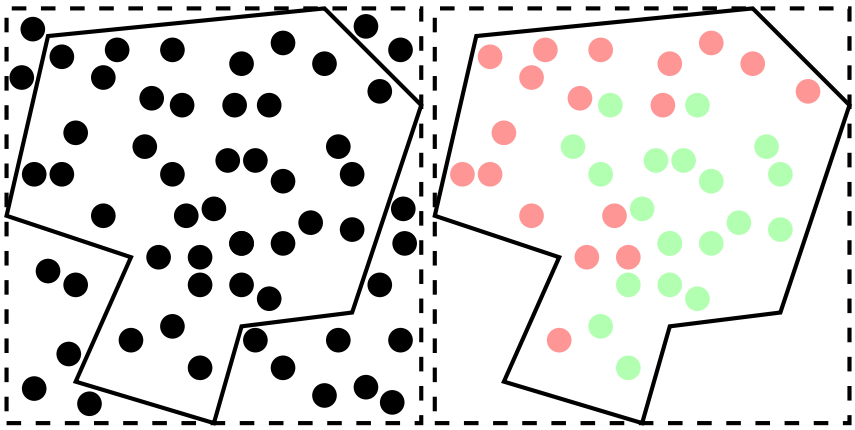

Figure 3. Points are sampled in the dashed bounding box (left), then design points outside the box are removed and the remaining ones are marked as good and bad (right).

3.2. **Trim Polygons.** In order to find the good solution space, a polygon needs to be trimmed such that it contains no more bad sample points. This is done by successively removing bad sample points from the polygon. To accomplish this, a bad sample point is specified. A good sample point is chosen (see Figure 4, top left) and a triangle out of this good sample point and two neighbouring vertices is formed, such that the bad sample point

is contained in this triangle (see Figure 4, top right). Then, the vertices are moved towards the good sample point until the bad sample point lies on the edge of the polygon (see Figure 4, bottom left and right). Thus, the bad sample point lies no longer in the polygon. This procedure is then repeated for each good sample point, yielding multiple polygons that are differently trimmed. From those, the best polygon (according to the quality measures in Subsection 3.3) is chosen as the new, trimmed polygon. Then, this procedure is repeated again for all bad sample points remaining in the trimmed polygon.

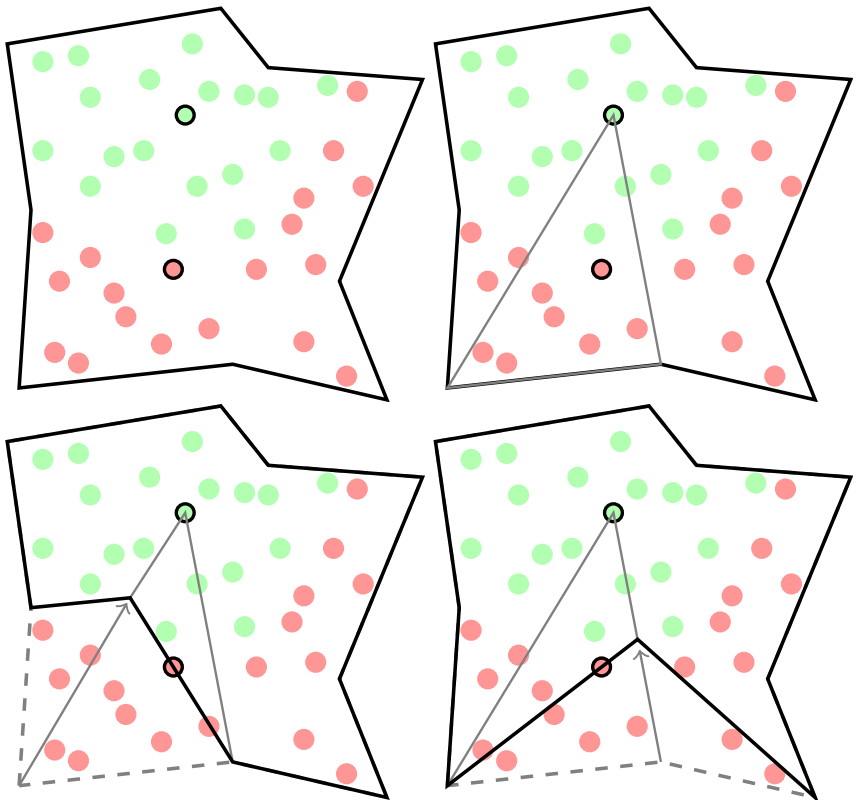

FIGURE 4. From a good and a bad point (top left) a triangle is constructed (top right) and the vertices are moved toward the good point for two possible polygons (bottom left and right).

The details of this procedure can be found in Algorithm 1. It requires a polygon P with vertices $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}$, a good point $\boldsymbol{x}^{\text{good}}$ and a bad point $\boldsymbol{x}^{\text{bad}}$ as inputs (line 1). For each vertex $\boldsymbol{v}^{(k)}$, it is checked whether the bad point lies within the convex hull of $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{v}^{(k)}$ and $\boldsymbol{v}^{(k+1)}$, which is exactly the triangle formed by those points (lines 3 and 4). If $\boldsymbol{x}^{\text{bad}}$ does lie within the triangle, the polygon is trimmed as explained above by Algorithm 2

Algorithm 1 `trim_polygon`: Trim a polygon, keeping as many good sample points as possible.

1: **Input:** P, $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{x}^{\text{bad}}$
2: **Output:** P

3: **for all** $\boldsymbol{v}^{(k)} \in P$ **do**
4: **if** $\boldsymbol{x}^{\text{bad}} \in \text{conv}(\{\boldsymbol{x}^{\text{good}}, \boldsymbol{v}^{(k)}, \boldsymbol{v}^{(k+1)}\})$ **then**
5: $P_1 \leftarrow \texttt{trim\_triangle}(P, \boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \boldsymbol{v}^{(k)}, \boldsymbol{v}^{(k+1)})$
6: $P_2 \leftarrow \texttt{trim\_triangle}(P, \boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \boldsymbol{v}^{(k+1)}, \boldsymbol{v}^{(k)})$
7: $P^{(k)} \leftarrow \texttt{evaluate}(P_1, P_2)$
8: **end if**
9: **end for**
10: $P \leftarrow \texttt{evaluate}(\{P^{(k)}\}_{k=1}^{N})$

(lines 5 and 6). The two possible outcomes are evaluated with the quality measures from Section 3.3 and the better one is kept (line 7). Finally, the best polygon $P^{(k)}$ is chosen as output in line 10.

Algorithm 2 takes a polygon P, a good point $\boldsymbol{x}^{\text{good}}$, a bad point $\boldsymbol{x}^{\text{bad}}$, and two neighbouring vertices $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ as input arguments (line 1). It trims the triangle formed by $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ by moving the edge between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ such that it lies on $\boldsymbol{x}^{\text{bad}}$. At first, the edges of the triangle are initialized (lines 3 and 4). Then, in line 5, the system

$$[\mathbf{e}_1, -\mathbf{e}_2] \cdot \mathbf{t} = \boldsymbol{v}_2 - \boldsymbol{v}_1$$

is solved and the value t_1 is used to determine how far the vertex $\boldsymbol{v}_1$ has to be moved (line 6). Finally, the corresponding vertex in the polygon is updated (line 7).

Algorithm 2 `trim_triangle`: Trim a triangle inside a polygon.

1: **Input:** P, $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{x}^{\text{bad}}$, $\boldsymbol{v}_1$, $\boldsymbol{v}_2$
2: **Output:** P

3: $\mathbf{e}_1 \leftarrow \boldsymbol{x}^{\text{good}} - \boldsymbol{v}_1$
4: $\mathbf{e}_2 \leftarrow \boldsymbol{x}^{\text{bad}} - \boldsymbol{v}_2$
5: Solve $[\mathbf{e}_1, -\mathbf{e}_2] \cdot \mathbf{t} = \boldsymbol{v}_2 - \boldsymbol{v}_1$
6: $\boldsymbol{v}_1 \leftarrow \boldsymbol{v}_1 + t_1 \cdot \mathbf{e}_1$
7: $P \leftarrow \texttt{update}(P, \boldsymbol{v}_1)$

3.3. **Evaluation of Polygons.** As multiple trimmed polygons are obtained at several steps of the optimization algorithm, the best polygon has to be chosen from among those polygons. For this purpose, quality measures for the polygons need to be introduced. A polygon not fulfilling one of these measures is immediately rejected and not used in the algorithm further. The polygons are rated as follows:

Minimum number of self-intersections. Each polygon should be free of self-intersections. Self-intersections lead to unwanted behaviour of the algorithms. It is not clear how to trim a polygon with self-intersections, and multiple self-intersections overlaying each other obscure what the inside of the polygon is. Thus, polygons having no self-intersections are preferred over polygons with self-intersections (see Figure 5).

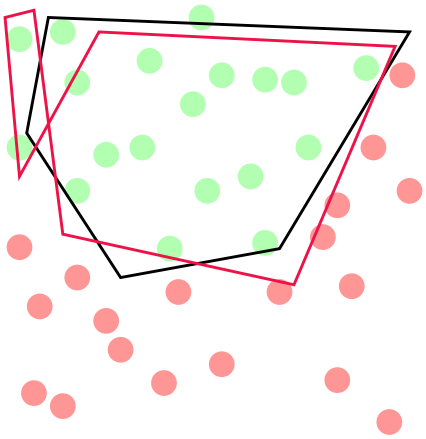

FIGURE 5. The black polygon is preferred over the red polygon because it has no self-intersections.

Minimum/maximum size of angles. Polygons with very small angles or very large angles form spikes, see Figure 6. When a spike is trimmed, it is very likely that a self-intersection is induced. Additionally, there is only a small chance for a point to be sampled within a spike, which in turn means that the spike will not be removed in a trimming step, making the vertex in the corner of the spike redundant. For these reasons, polytopes with no or only a few spikes are preferred. For a fixed threshold angle α, polygons which satisfy $\alpha < \phi < 2\pi - \alpha$ for as many vertex angles ϕ as possible (see Figure 6) are favored over others.

Maximum number of good points. Finally, after rejecting all polygons with a bad shape, the size of the good design space is considered. Therefore, the numbers of good points within the polygons are compared and the polygon containing the most is chosen. If that polygon is not unique because several polygons contain the same highest number of points, then one from among those is chosen at random (see Figure 7).

3.4. **Remove Spikes.** After trimming and evaluating the polygon, it might still contain spikes. If this is the case, i.e., if there are vertices whose angles ϕ violate the condition $\alpha < \phi < 2\pi - \alpha$, then these vertices are relocated, as seen in Figure 8. After this step, the polygon has no spikes any more while losing not too much of its volume.

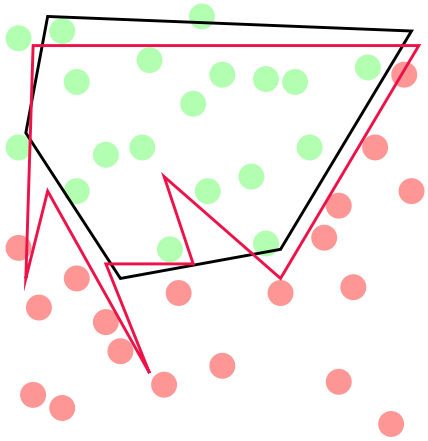

FIGURE 6. The black polygon is preferred over the red polygon because it has less spikes.

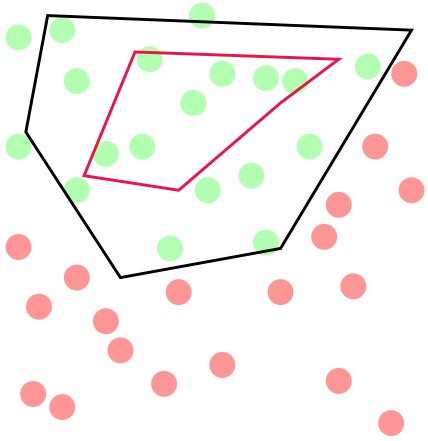

FIGURE 7. The black polygon is preferred over the red polygon because it contains more good points.

3.5. **Relocate Vertices.** A further manipulation step consists of relocating vertices. The idea behind this step is to avoid degeneration of the polygon, especially to avoid regions where vertices form clusters. This reduces the risk of a polygon developing new spikes.

The strategy of the relocation is as follows: The shortest edge of the polygon is removed by replacing its endpoints by the midpoint of the edge. Hence, one vertex is removed from the polygon. To keep the number of vertices constant, a new vertex is placed at the midpoint of the longest edge of the polygon (see Figure 9).

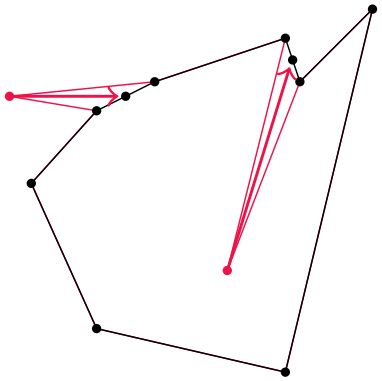

FIGURE 8. Relocating vertices in order to remove spikes (red) from a polygon.

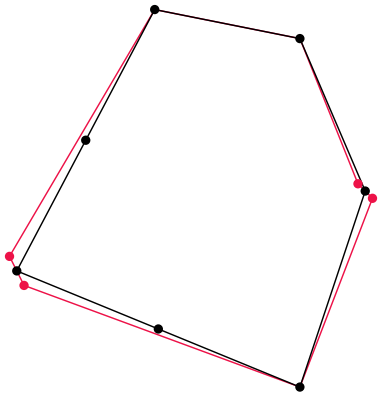

FIGURE 9. Two short edges (red) are removed from a polygon, then two vertices are added on the longest edges.

3.6. **Grow Polygon.** In this step, the polygon is grown in all directions. This allows the polygon to extend into regions of good design space. Every vertex of the polygon is moved by the same factor g along its outward pointing angle bisector (see Figure 10). The vector of the angle bisector is normalized to 1.

3.7. **Remove Self-Intersections.** Sometimes, the trimming, growing, and relocating steps might introduce self-intersections of the polygon, despite the quality measures trying to prevent this. Therefore, an algorithm is presented that removes those self-intersections. It removes the interior self-intersections by finding the hull of the polygon, whose vertices coincide with all vertices of the original polygon which do not lie within that polygon, compare Figure 11 for an illustration of this procedure. This part of the algorithm is based on the *Graham scan method*, which finds the convex hull of a finite set of points, see [15]. Then, if the polygon consists of multiple

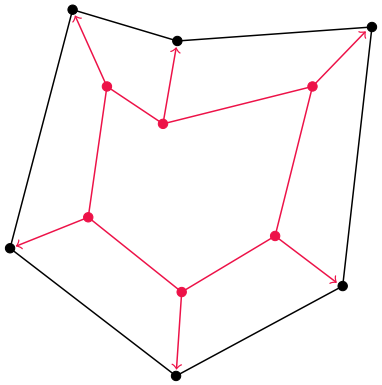

FIGURE 10. Growing a polygon.

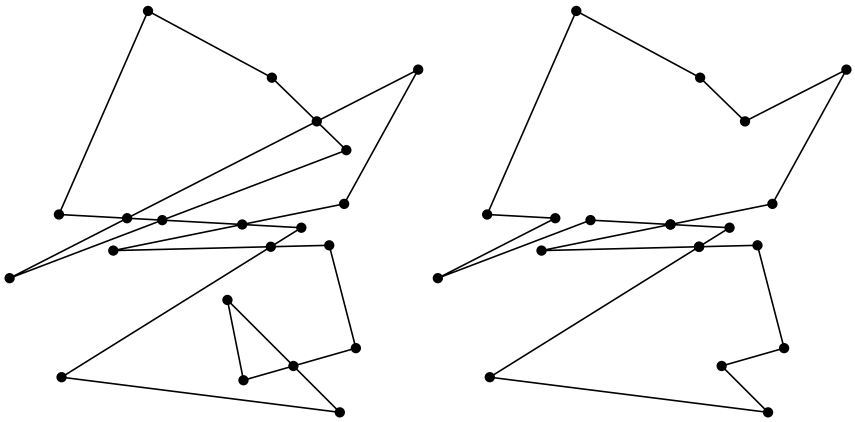

FIGURE 11. A polygon with self-intersections (left) and its counterpart without self-intersections (right).

connected components, the largest of these connected components is chosen as the new polygon and all the smaller components are removed. Afterwards, vertices are added or removed to maintain the total number of vertices.

In detail, the algorithm consists of the following steps:

Starting with the vertex that has the smallest y-coordinate (it is for sure a vertex of the new polygon), those line segments are considered that directly connect the vertex to other vertices and intersections. From these line segments, the one that encloses the smallest angle with the x-axis and its endpoint is chosen as an edge of the new polygon. This procedure is repeated from the endpoint of that edge, compare Figure 12.

When the algorithm arrives at the starting vertex again, it has found the hull of the polygon and terminates, see Figure 13.

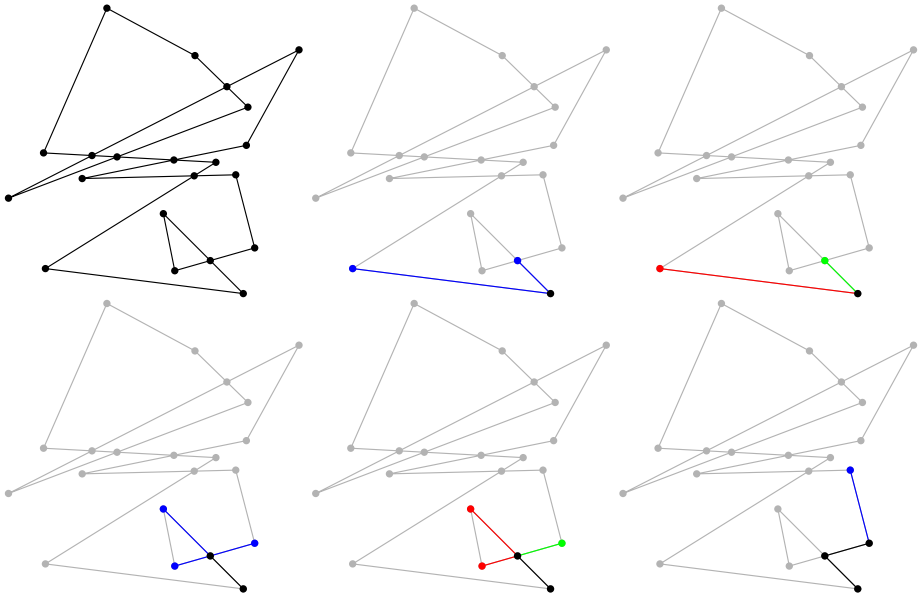

FIGURE 12. Finding the polygon hull. The possible line segments to choose from in each step are blue, the discarded line segments are red, and the chosen segment is green.

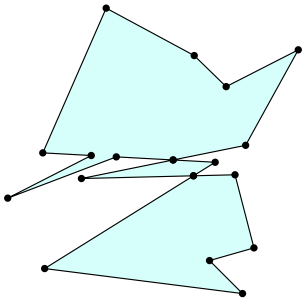

FIGURE 13. Final hull of the polygon.

Nonetheless, the polygon might still consist of multiple different connected components. Thus, the polygon hull algorithm is implemented such that the list of vertices of the hull is given as an output. All vertices that appear more than once in the list are points where at least two different connected components are touching. If the polygon consists of more than two connected components, this information can be used to recursively find all connected components of the polygon hull. Then, the largest of the connected components is chosen as the new polygon. Finally, vertices are added like in the relocate vertices step (cf. Section 3.5) to regain the prescribed amount of vertices. We refer to Figure 14 for an illustration.

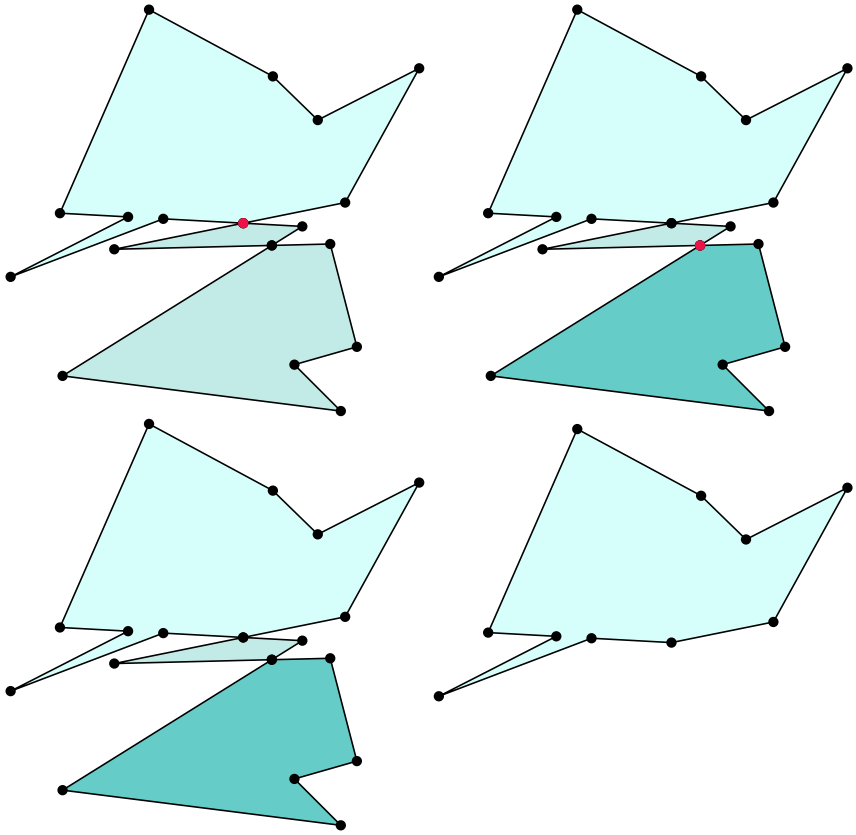

FIGURE 14. Finding and choosing the largest connected component of the polygon's hull.

4. Polytope Optimization Algorithm

The steps in the polytope optimization algorithm are very similar to those of the box optimization algorithm and the rotated box optimization algorithm (see [16]). An overview of the most important steps of the polytype optimization algorithm can be found in Figure 15.

The initial polytope is usually given. If no specific polytope is given, one can use genetic algorithms [14] to find points around which a polytope could be constructed. All polygons $P_{i,j}$ have a fixed number N of vertices, and, in the algorithm, each polygon is represented by an ordered sequence of vertices, that is

$$P_{i,j} = \left\{ \boldsymbol{v}_{i,j}^{(1)}, \dots, \boldsymbol{v}_{i,j}^{(N)} \right\}.$$

4.1. **Exploration Phase.** In the *exploration phase*, those parts of the initial polytope are trimmed that contain bad design points. Then, the polytope is grown again. These steps are repeated n^{exp} times. This allows the

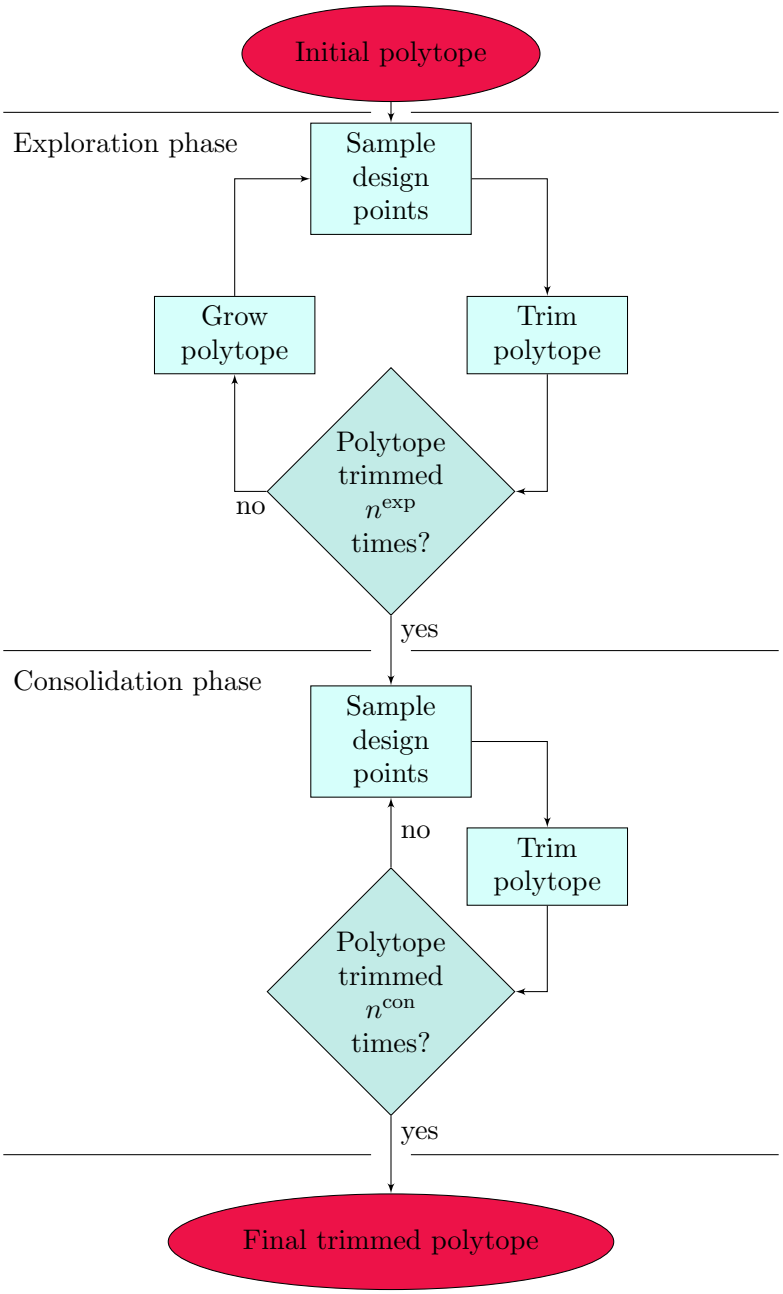

FIGURE 15. A flowchart for the polytope optimization algorithm.

polytope to move through the design space Ω in order to find a spot with a large volume of good design space. After going through all n^{exp} steps of the exploration phase, the algorithm switches to the *consolidation phase*.

4.1.1. *Sample Points.* For each sample point $\boldsymbol{x}$, all entries x_k, $k \in \mathcal{J}_{\text{int}}$, and x_i, x_j, $(i,j) \in \mathcal{J}_{\text{pair}}$, are sampled separately. The entries x_k can easily be drawn from the interval I_k. The entries (x_i, x_j) are obtained by sampling a point inside the polygon $P_{i,j}$ as described in Subsection 3.1. After the sample points are found, they are evaluated. All of the sample points $\boldsymbol{x}$ with an objective value $f(\boldsymbol{x}) \leq c$ are collected in the set

$$\mathcal{X}^{\text{good}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{good}}, \dots, {}_{(n^{\text{good}})}\boldsymbol{x}^{\text{good}} \right\},$$

and all of the sample points $\boldsymbol{x}$ with an objective value $f(\boldsymbol{x}) > c$ are collected in the set

$$\mathcal{X}^{\text{bad}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{bad}}, \dots, {}_{(n^{\text{bad}})}\boldsymbol{x}^{\text{bad}} \right\},$$

compare Figure 16. The sets are sorted by the objective values of the sample points, from highest to lowest.

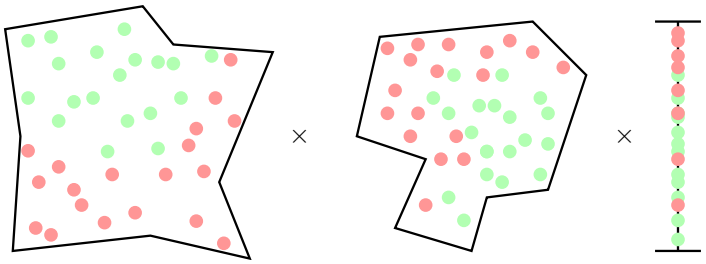

FIGURE 16. A polytope consisting of two polygons and an interval with good and bad sample points.

4.1.2. *Trim Polytope.* After the sampling step, the bad points are removed by trimming the polytope. The framework of this step is outlined in Algorithm 3. As input, a polytope Ω_{pol} and ordered sets of good sample points $\mathcal{X}^{\text{good}}$ and bad sample points $\mathcal{X}^{\text{bad}}$ (line 1) are required. The output (line 2) is a polytope Ω_{pol} that contains no more bad sample points.

Because the bad sample points have to be removed successively, a loop over the bad sample points is initialized in line 3. Since as many good sample points as possible should be kept, the good sample points are iterated and each sample point ${}_{(m)}\boldsymbol{x}^{\text{good}}$ is set as an anchor point once (line 4). For each anchor, the current bad sample point ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ is removed from the polytope such that at least the anchor point ${}_{(m)}\boldsymbol{x}^{\text{good}}$ remains within the polytope. The bad sample point ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ is removed by moving the boundary of an interval I_k or a polygon $P_{i,j}$ onto ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$, thereby trimming the polytope. Thus, all intervals I_k and polygons $P_{i,j}$ are iterated (see lines 5–10). For each interval I_k and polygon $P_{i,j}$, the boundary is moved onto ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ via the `trim_interval` and `trim_polygon` algorithms and the resulting polytope is stored in a new variable $\Omega^{(k)}$ or $\Omega_{i,j}$, respectively, leaving all other intervals

Algorithm 3 Polytope trimming: Trim the polytope, keeping as many good sample points as possible.

1: **Input:** Ω_{pol}, $\mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}$
2: **Output:** Ω_{pol}

3: **for** $\ell = 1, \ldots, n^{\text{bad}}$ **do**
4: **for** $m = 1, \ldots, n^{\text{good}}$ **do**
5: **for all** $k \in \mathcal{J}_{\text{int}}$ **do**
6: $\Omega^{(k)} \leftarrow \texttt{trim\_interval}(\Omega_{\text{pol}}, k, {}_{(m)}\boldsymbol{x}^{\text{good}}, {}_{(\ell)}\boldsymbol{x}^{\text{bad}})$
7: **end for**
8: **for all** $(i,j) \in \mathcal{J}_{\text{pair}}$ **do**
9: $\Omega_{i,j} \leftarrow \texttt{trim\_polygon}(\Omega_{\text{pol}}, (i,j), {}_{(m)}\boldsymbol{x}^{\text{good}}, {}_{(\ell)}\boldsymbol{x}^{\text{bad}})$
10: **end for**
11: $\Omega^{(\ell)} \leftarrow \texttt{evaluate}\left(\{\Omega^{(k)}\}_{k\in\mathcal{J}_{\text{int}}}, \{\Omega_{i,j}\}_{(i,j)\in\mathcal{J}_{\text{pair}}}\right)$
12: **end for**
13: $\Omega_{\text{pol}} \leftarrow \texttt{evaluate}(\{\Omega^{(\ell)}\}_{\ell=1}^{n^{\text{good}}})$
14: **end for**
15: $\Omega_{\text{pol}} \leftarrow \texttt{reshape}(\Omega_{\text{pol}})$

and polygons untouched. Note that the algorithm `trim_polygon` coincides with Algorithm 1 except for needing the coordinates (i,j) of the respective polygon $P_{i,j}$ as input arguments. The algorithm `trim_interval` operates on the interval I_k and simply relocates one of the end points onto the bad sample point such that the good sample point remains in the output interval.

Then, in line 11, the function `evaluate` is applied to all polytopes $\Omega^{(k)}$ and $\Omega_{i,j}$. It returns the result $\Omega^{(\ell)}$ that maximizes the quality measures, applied in the same order as listed in Subsection 3.3. The quality measures are modified for polytopes such that the polytope with the most polygons that fulfill the self-intersection and angle-size measures that also contains as many good design points as possible is chosen as the optimum.

After every good point has been set as anchor once, the polytopes $\Omega^{(\ell)}$ (line 13) are evaluated and the best of them is used to replace Ω_{pol}. Following this, the next iteration of the loop starts, where the next bad design point is removed. The polygon trimming is completed when all of the bad points are removed.

In order to avoid degenerate polytopes, finally spikes are removed and vertices are relocated by the function `reshape` in line 15. This function consists of the operations *remove spikes* (see Subsection 3.4) and *relocate vertices* (see Subsection 3.5), which are applied to each polygon $P_{i,j}$ individually as explained in Subsections 3.4 and 3.5, respectively.

4.1.3. *Grow Polytope.* The polytope is grown as the final operation of a single step of the exploration phase. To this end, the end points a_k and b_k

of each interval I_k are moved by a factor $g^{(\ell)}$ in order to grow the polytope in each dimension k. Each polygon $P_{i,j}$ is grown by the factor $g^{(\ell)}$ as explained in Subsection 3.6.

The factor $g^{(\ell)}$ is the growth rate in the ℓ-th step of the exploration phase. It can either be constant or dynamic. A constant growth factor means that $g^{(0)} = g^{(1)} = \cdots = g^{(n^{\text{exp}})}$. A dynamic growth factor depends on the number of good design points and the growth rate of the previous step (see [14]):

$$g^{(\ell)} = \frac{a_\ell^{\text{good}}}{a^{\text{target}}} g^{(\ell-1)}.$$

Here, $a_\ell^{\text{good}} = n^{\text{good}}/(n^{\text{good}} + n^{\text{bad}})$ is the percentage of good points in the ℓ-th exploration step before trimming the polytope and a^{target} is the desired percentage of good design points inside the polytope during the exploration phase.

The growth factor is large when many good design points have been found in the sampling step before trimming the polytope. This indicates that the polytope lies in a region with good design space and it could gain potentially more good design space by growing. The growth factor is small when the polytope contains many bad design points, because this implies that the polytope grew out of the good design space into bad design space. Thus, the growth rate should be kept small in order to ensure that the polytope, after having been trimmed, can probe for the potential border between the good and the bad design space.

4.2. **Consolidation Phase.** Having completed the exploration phase, the candidate polytope might still contain some bad design space. The goal of the *consolidation phase* is to remove as much bad design space as possible. Thus, one step of the consolidation phase consists of the execution of *sample points* (see Subsection 4.1.1) and *trim polytope* (see Subsection 4.1.2). During this phase, the polytope is no longer grown. The consolidation phase is terminated after either a fixed number of n^{con} steps or when no bad design points have been sampled three times in series. The resulting polytope is returned as final output for the polytope optimization algorithm.

5. Numerical Experiments

5.1. **Problem 1: 2d Polygon.** We consider a first simple two-dimensional test example. For

$$\mathbf{A} = \begin{bmatrix} 1/8 & 1/4 \\ 4/17 & 2/17 \\ -1/2 & 1/2 \\ -1/2 & -1/3 \\ -1/3 & -2/3 \\ 1 & -3/2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix},$$

we consider the problem of finding $\boldsymbol{x} \in \Omega_{\text{ds}} = [0, 4]^2$ such that

$$f(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} - \mathbf{b} \leq \mathbf{0}.$$

The good design space is a two-dimensional six-sided polygon (compare Figure 17).

Algorithm	Mean Absolute Volume	Mean Normalized Volume
Box Optimization	2.35	0.15
Rotated Box Optimization	2.99	0.19
Polytope Optimization	4.28	0.27

TABLE 1. Results of the different optimization algorithms in case of Problem 1.

The problem under consideration shows in a simple manner how the polytope optimization works. Especially, it allows for an easy comparison between the box optimization, the rotated box optimization, and the polytope optimization algorithms. Each of these algorithms has been applied 100 times to the objective function f. Every time, the number of steps in the exploration and the consolidation phase is set to $n^{\text{exp}} = n^{\text{con}} = 100$. Additionally, in every step, 100 design points are sampled. The growth rate is dynamic, with $a^{\text{target}} = 0.8$ and $g^{(0)} = 0.05$. The polytope optimization has been performed with polygons that have 10 vertices, where the required minimum size of angles is $\alpha = 20°$ and the vertex relocation takes place in every tenth step of the exploration phase. The results can be found in Table 5.1.

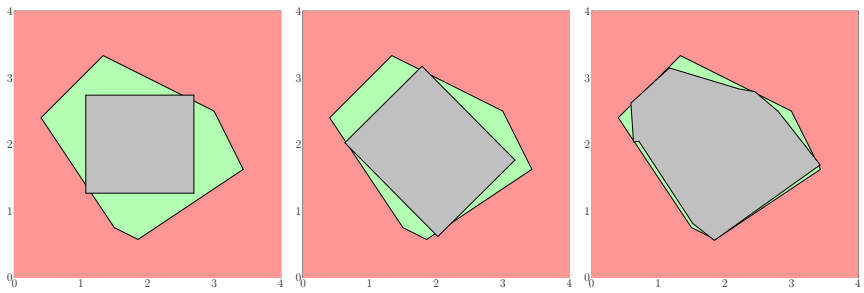

FIGURE 17. Problem 1: An axis-parallel solution space (left), a rotated solution space (middle) and a polygonal solution space (right) of normalized volumes 0.15, 0.2 and 0.27, respectively. The green color indicates the good solution space.

As one might have expected, the polytope optimization yields the highest average volume for this problem, with 80% more volume than the box

optimization and 42% more volume than the rotated box optimization. Examples for the solution spaces found by the different optimization algorithms are given in Figure 17.

5.2. **Problem 2: 4d Rosenbrock Function.** The Rosenbrock function is a popular test function for optimization techniques. For $\boldsymbol{x} \in \Omega_{\text{ds}} = ([-2,2] \times [-2,3])^{d/2}$, where d is even, it is given by the formula

$$f(\boldsymbol{x}) = \sum_{i=1}^{d/2} (1 - x_{2i-1})^2 + 100(x_{2i} - x_{2i-1}^2)^2.$$

The rotated box optimization and the polytope optimization are applied 100 times on the problem, with $d = 4$ and $c = 120$. The 2d-spaces for the rotated box optimization and the polytope optimization are set to $\Omega_{1,2} = \Omega_{3,4} = [-2,2] \times [-2,3]$. The parameters are set as in Problem 1, except that, for the dynamic growth rate , $a^{\text{target}} = 0.6$ and the minimum angle for inside the polygons is set to $10°$. The mean absolute volume of the rotated box optimization is 3.26 (mean normalized volume: 0.0082) and the mean absolute volume of the polytope optimization is 16.4 (mean normalized volume: 0.041). This means that solution spaces found by the polytope optimization have approximately 500% more volume than those found by the rotated box optimization.

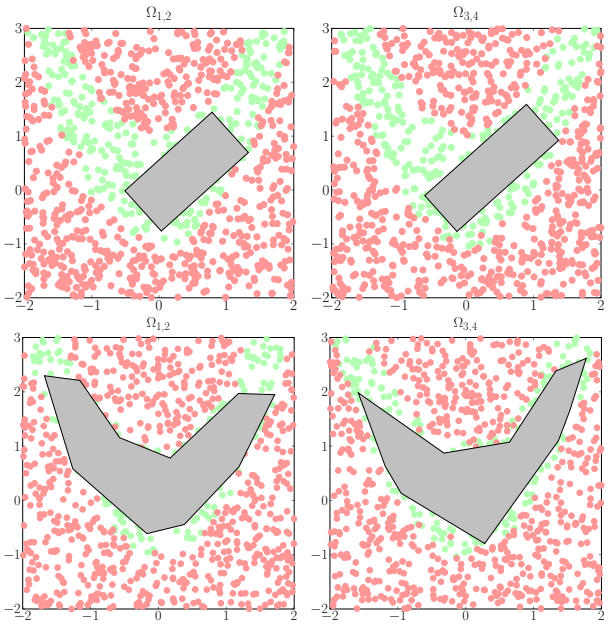

FIGURE 18. Problem 2: A rotated box with volume 3.2 and a polytope with volume 16.77.

In Figure 18, a rotated box with absolute volume 3.2 and a polytope with volume 16.77 are plotted. Note that this visualization is different than before. On each 2d-space $\Omega_{i,j}$, 1000 design points $\boldsymbol{x} = (x_1, \dots, x_4)$ are sampled with $x_k \in \Omega_{\text{box}}$ or $x_k \in \Omega_{\text{pol}}$, respectively, for $k \neq i, j$ and $x_k \in \Omega_{i,j}$ for $k = i, j$. This means that every design point is inside of Ω_{pol}, except for the coordinates x_i and x_j, which may be distributed anywhere on the 2d-space $\Omega_{i,j}$. In a certain way, this illustrates the region around Ω_{pol} from the "inside" of Ω_{pol}. This visualization makes clear that the results are reasonable and not much more good design space could be gained by the polytope optimization, Ω_{pol} fills out most of the U-shaped good design space, whereas the rotated box only fills one side on each 2d-space.

5.3. **Problem 3: Application to an Optimal Control Problem.** Consider the following problem: Five heat sources have to be designed such that they keep the temperature in a control volume on a given constant level. Each heat source has a fixed position $\boldsymbol{x}_i = (x_{i,1}, x_{i,2})$ in the control volume and a circular shape with radius r_i. The temperature at the i-th heat source is given by the constant factor t_i. The distribution of the heat emitted by the heat sources throughout the control volume is modelled by the *steady-state heat equation* (compare [28] for example)

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= g_{\boldsymbol{r},\boldsymbol{t}}(\boldsymbol{x}), & \boldsymbol{x} &\in D := (0,1)^2, \\ u(\boldsymbol{x}) &= 0, & \boldsymbol{x} &\in \Gamma := \partial D, \end{aligned} \tag{5.1}$$

where

$$g_{\boldsymbol{r},\boldsymbol{t}}(\boldsymbol{x}) = \sum_{i=1}^{5} t_i \cdot \mathcal{X}_{B_{r_i}(\boldsymbol{x}_i)}(\boldsymbol{x}).$$

Here, $\mathcal{X}_B$ is the characteristic function

$$\mathcal{X}_B(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in B, \\ 0, & \text{else}, \end{cases}$$

and $B_{r_i}(\boldsymbol{x}_i)$ denotes the ball with radius r_i around the center $\boldsymbol{x}_i$. The positions of the centers $\boldsymbol{x}_i$ are given by

$$\begin{gathered} \boldsymbol{x}_1 = (0.15, 0.4), \quad \boldsymbol{x}_2 = (0.45, 0.9), \quad \boldsymbol{x}_3 = (0.87, 0.7), \\ \boldsymbol{x}_4 = (0.88, 0.25), \quad \boldsymbol{x}_5 = (0.5, 0.3). \end{gathered}$$

We intend to determine the variables $\boldsymbol{r}$ and $\boldsymbol{t}$ such that the maximum deviation from the desired temperature, i.e.

$$f(\boldsymbol{r}, \boldsymbol{t}) = \|u^\star_{\boldsymbol{r},\boldsymbol{t}} - u_d\|_{L^\infty(K)}$$

is minimized. Here, $u_d = 0.5$ is the desired constant temperature and $K := [0.3, 0.7]^2 \subset D$ the region inside the control volume where that temperature should be close to u_d. The problem under consideration is an optimal control problem, where $\boldsymbol{r}$ and $\boldsymbol{t}$ are the *control variables* to be determined such that they minimize the *cost function* f, see [28] for example.

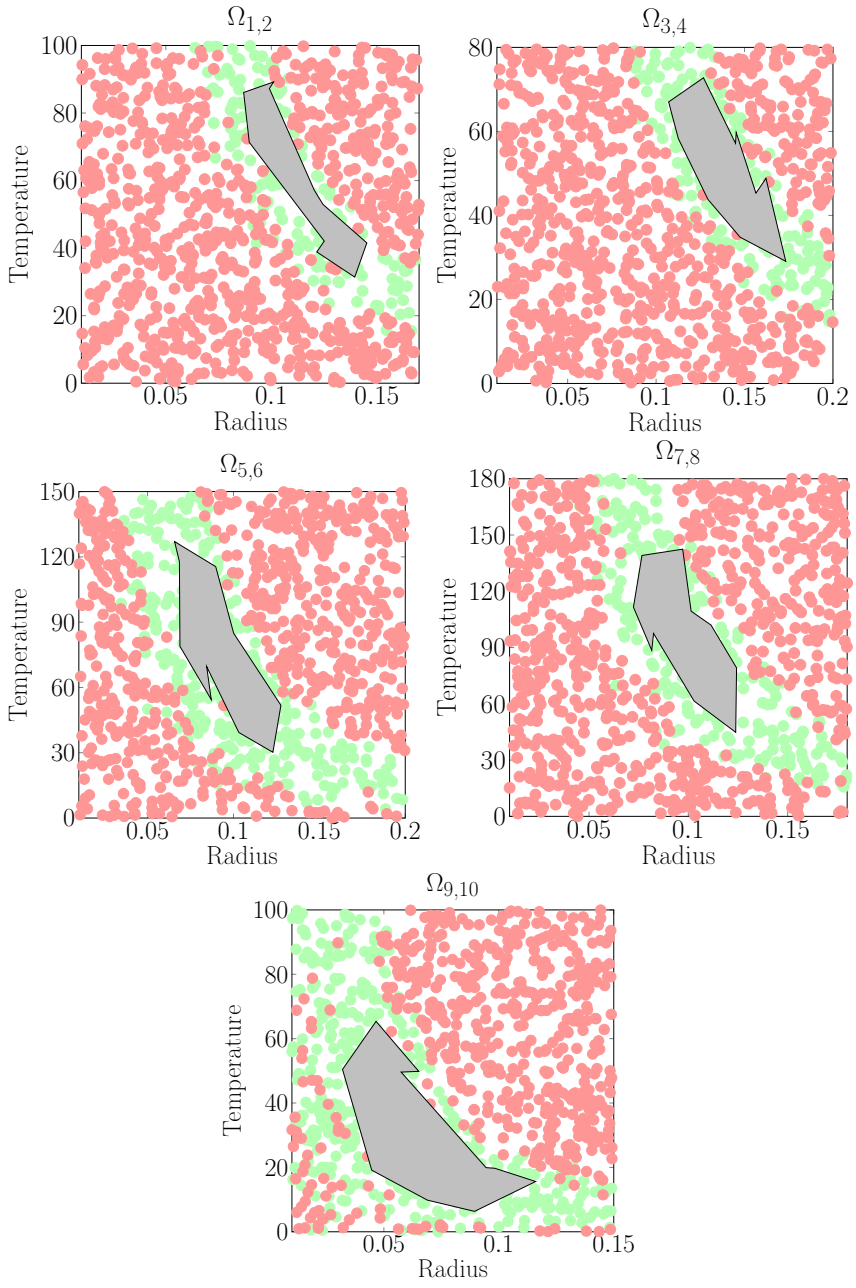

FIGURE 19. Problem 3: A solution space with volume 11.88.

In the context of the polytope optimization, f is the objective function and $\boldsymbol{r}$ and $\boldsymbol{t}$ are the design variables, where we choose $\Omega_{\text{ds}} := \Omega_{\boldsymbol{r}} \times \Omega_{\boldsymbol{t}}$ as design

space with

$$\Omega_{\boldsymbol{r}} = [0.01, 0.17] \times [0.01, 0.2] \times [0.01, 0.2] \times [0.01, 0.18] \times [0.01, 0.15],$$
$$\Omega_{\boldsymbol{t}} = [0, 100] \times [0, 80] \times [0, 150] \times [0, 180] \times [0, 100].$$

The radius and temperature of each heat source are coupled by a 2d-space such that $\Omega_{1,2} = [0.01, 0.17] \times [0, 100]$, $\Omega_{3,4} = [0.01, 0.2] \times [0, 80]$, $\Omega_{5,6} = [0.01, 0.2] \times [0, 150]$, $\Omega_{7,8} = [0.01, 0.18] \times [0, 180]$, and $\Omega_{9,10} = [0.01, 0.15] \times [0, 100]$.

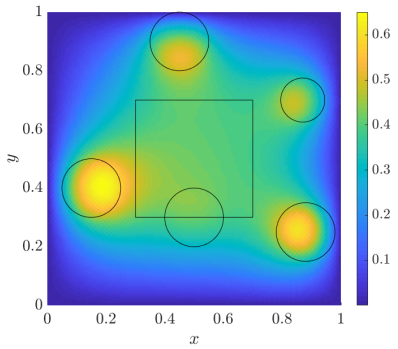

FIGURE 20. Problem 3: A solution of the heat equation represented by a design point taken from within the polytope seen in Figure 19. The region K is marked with a black square and the radii of the heat sources are marked with black circles.

Again, the algorithm is applied 100 times with 100 steps in the exploration and consolidation phases and 100 sampled design points in each step. Each polygon on a 2d-space has 10 vertices, the minimum angle is 10° and the vertices are relocated every 10 steps. The growth rate is dynamic with $a_{\text{target}} = 0.6$ and $g^{(0)} = 0.05$. Moreover, as critical value, we choose $c = 0.15$, which means that the temperature generated by the heat sources is allowed to deviate by up to 30% from the desired temperature.

The resulting mean absolute volume of the solution spaces is 11.37 and the mean normalized volume is $1.37 \cdot 10^{-6}$. Figure 19 shows a polytope with absolute volume 11.88, plotted in the same way as Figure 18. It fills most of the pocket of good design space it has found. A solution of (5.1) with a design taken from within that polytope is plotted in Figure 20. Figure 21 shows each 2d-space as a heat map of the 100 solution spaces. The brighter a region, the more solution spaces cover that region. The picture suggests that they usually stay within the same region, so it can be concluded that the algorithm delivers robust results.

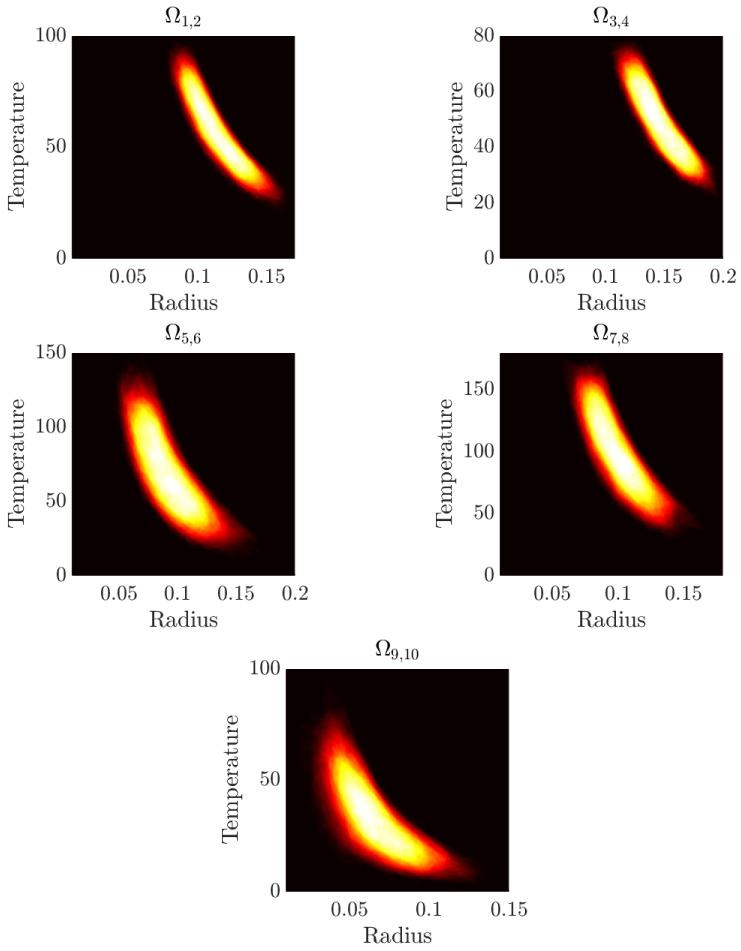

FIGURE 21. Problem 3: Heat map for 100 solution spaces.

6. Conclusion

The algorithm presented in this article utilizes the concept of 2d-spaces, introduced in [8], to form a coupling between the coordinates. It replaces the box-shaped solution space on each 2d-space by a polygonal solution space, whose product results in a high-dimensional polytope. Numerical results confirm that this setting allows the algorithm to find solution spaces of larger volume than those found by its predecessors from [16] and [32] while the cost, i.e., the number of sample points, is the same.

References

[1] H. Beyer and B. Sendhoff. Robust optimization. A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196:3190–3218, 2007.

[2] M. Beer and M. Liebscher. Designing robust structures. A nonlinear simulation based approach. *Computers and Structures*, 86:1102–1112, 2007.
[3] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics and Mathematical Programming Society, Philadelphia, 2001.
[4] J.H. Conway, H. Burgiel, and C. Goodman-Strauss. *The Symmetries of Things*. A.K. Peters, Wellesley, 2008.
[5] P.K. Das, D. Faulkner, and Y. Pu. A strategy for reliability-based optimization. *Engineering Structures*, 19(3):276–282, 1997.
[6] M. Eichstetter, S. Müller, and M. Zimmermann. Product family design with solution spaces. *Journal of Mechanical Design*, 137(12):121401, 2015.
[7] M. Eichstetter, C. Redeker, S. Müller, P. Kvasnicka, and M. Zimmermann. Solution Spaces for Damper Design in Vehicle Dynamics. 5th International Munich Chassis Symposium 2014. chassistech plus. Proceedings, Springer Fachmedien Wiesbaden, pp. 107–132, 2015.
[8] S. Erschen, F. Duddeck, M. Gerdts, and M. Zimmermann. On the optimal decomposition of high-dimensional solution spaces of complex systems. *ASME Journal of Risk and Uncertainty in Engineering Systems, Part B*, 4(2):021008, 2017.
[9] J. Fender, F. Duddeck, and M. Zimmermann. Direct computation of solution spaces for crash design. *Structural and Multidisciplinary Optimization*, 55(5):1787–1796, 2017.
[10] G. Fung, S. Sandilya, and R.B. Rao. Rule Extraction from Linear Support Vector Machines. *KDD '05 Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ACM, New York, pp. 32–40, 2005.
[11] L. Graff. *A stochastic algorithm for the identification of solution spaces in high-dimensional design spaces*. PhD thesis, Faculty of Science, University of Basel, 2013.
[12] L. Graff, J. Fender, H. Harbrecht, and M. Zimmermann. Identifying key parameters for design improvement in high-dimensional systems with uncertainty. *Journal of Mechanical Design*, 136(4):041007, 2014.
[13] W. Graf, M. Götz, and M. Kaliske. Computing permissible design spaces under consideration of functional responses. *Advances in Engineering Software*, 117:95–106, 2018.
[14] L. Graff, H. Harbrecht, and M. Zimmermann. On the computation of solution spaces in high dimensions. *Structural and Multidisciplinary Optimization*, 54(4):811–829, 2016.
[15] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.
[16] H. Harbrecht, D. Tröndle, and M. Zimmermann. A sampling-based optimization algorithm for solution spaces with pair-wise coupled design variables. *Structural and Multidisciplinary Optimization*, 60(2):501–512, 2019.
[17] C. Haskins. *Systems Engineering Handbook v.3*. INCOSE, San Diego, CA, 2006
[18] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20:131–144, 2001.
[19] J.R.R.A. Martins and A.B. Lambe. Multidisciplinary Design Optimization: Survey of Architectures. *AIAA Journal* 51(9):2049–2075, 2013.
[20] S. Königs and M. Zimmermann. Resolving Conflicts of Goals in Complex Design Processes. Application to the Design of Engine Mounts. 7th International Munich Chassis Symposium 2016. Proceedings. Springer Fachmedien Wiesbaden, pp. 125–141, 2017.
[21] J.-D. Korus, P. Karg, G.R. Pilar, C. Schütz, M. Zimmermann, and S. Müller. Robust design of a complex, perturbed lateral control system for automated driving. *IFAC-PapersOnLine*, 52(8):1–6, 2019.

[22] J.-D. Korus, G.R. Pilar, C. Schütz, M. Zimmermann, and S. Müller. Top-down development of controllers for highly automated driving using solution spaces. 9th International Munich Chassis Symposium 2018. Proceedings. Springer Fachmedien Wiesbaden, pp. 325–342, 2018.
[23] M. Münster, M. Lehner, D.J. Rixen, and M. Zimmermann. Vehicle steering design using solution spaces for decoupled dynamical subsystems. 26th Conference on Noise and Vibration Engineering ISMA 2014. Proceedings vol. 26, pp. 279–288, 2014.
[24] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 2006.
[25] R. Rackwitz. Reliability analysis. A review and some perspectives. *Structural Safety*, 23(4):365–395, 2001.
[26] C.M. Rocco, J.A. Moreno, and N. Carrasquero. Robust design using a hybrid-cellular–evolutionary and interval–arithmetic approach: a reliability application. *Reliability Engineering and System Safety*, 79:149–159, 2003.
[27] A. Saltelli, K. Chan, and E.M. Scott. *Sensitivity analysis*. Wiley, New York, 2000.
[28] F. Tröltzsch. *Optimale Steuerung partieller Differentialgleichungen*. Vieweg + Teubner, Wiesbaden, 2009.
[29] M. Vogt, F. Duddeck, M. Wahle, and M. Zimmermann. Optimizing tolerance to uncertainty in systems design with early-and late-decision variables. *IMA Journal of Management Mathematics*, 30(3):269–280, 2019.
[30] M. Zimmermann and M. Wahle. Solution spaces for vehicle concepts and architectures. Proceedings of the 24th Aachen Colloquium of Automobile and Engine Technology 2015, Aachen, Germany, pp. 689–698.
[31] M. Zimmermann, S. Königs, C. Niemeyer, J. Fender, C. Zeherbauer, R. Vitale, and M. Wahle. On the design of large systems subject to uncertainty. *Journal of Engineering Design*, 28(4):233–254, 2017.
[32] M. Zimmermann and J.E. von Hoessle. Computing solution spaces for robust design. *International Journal for Numerical Methods in Engineering*, 94(3):290–307, 2013.

Helmut Harbrecht, Dennis Tröndle, Departement Mathematik und Informatik, Universität Basel, Spiegelgasse 1, 4051 Basel, Schweiz.

Markus Zimmermann, Lehrstuhl für Produktentwicklung und Leichtbau, Technische Universität München, Boltzmannstr. 15, 85748 Garching, Deutschland.

LATEST PREPRINTS

No.	Author: *Title*
2018-02	**F. Ghiraldin and X. Lamy** *Optimal Besov differentiability for entropy solutions of the eikonal equation*
2018-03	**H. Harbrecht and M. Schmidlin** *Multilevel quadrature for elliptic problems on random domains by the coupling of FEM and BEM*
2018-04	**M. Bugeanu and H. Harbrecht** *Parametric representation of molecular surfaces*
2018-05	**A. Abdulle, M. J. Grote and O. Jecker** *Finite element heterogeneous multiscale method for Elastic Waves in Heterogeneous Media*
2018-06	**M. J. Grote and J. H. Tang** *On controllability methods for the Helmholtz equation*
2018-07	**H. Harbrecht and M. Moor** *Wavelet boundary element methods — Adaptivity and goal-oriented error estimation*
2018-08	**P. Balazs and H. Harbrecht** *Frames for the solution of operator equations in Hilbert spaces with fixed dual pairing*
2018-09	**R. Brügger, R. Croce and H. Harbrecht** *Solving a Bernoulli type free boundary problem with random diffusion*
2018-10	**J. Dölz, H. Harbrecht and M. D. Multerer** *On the best approximation of the hierarchical matrix product*
2018-11	**H. Harbrecht and P. Zaspel** *A scalable $\mathcal{H}$-matrix approach for the solution of boundary integral equations on multi-GPU clusters*
2018-12	**H. Harbrecht, N. Ilić and M. D. Multerer** *Acoustic scattering in case of random obstacles*
2018-13	**D. H. Baffet, M. J. Grote, S. Imperiale and M. Kachanovska** *Energy decay and stability of a perfectly matched layer for the wave equation*
2018-14	**D. Baffet and M. J. Grote** *On wave splitting, source separation and echo removal with absorbing boundary conditions*

LATEST PREPRINTS

No.	Author: *Title*
2019-01	**M. Graff, M. J. Grote, F. Nataf and F. Assous** *How to solve inverse scattering problems without knowing the source term: a three-step strategy*
2019-02	**Z. Gao and P. Habegger** *Heights in families of abelian varieties and the geometric Bogomolov conjecture*
2019-03	**M. J. Grote, F. Nataf, J. H. Tang and P.-H. Tournier** *Parallel Controllability Methods For the Helmholtz Equation*
2019-04	**G. Ciampa, G. Crippa and S. Spirito** *On smooth approximations of rough vector fields and the selection of flows*
2019-05	**M. Colombo, G. Crippa, M. Graff and L. V. Spinolo** *Recent results on the singular local limit for nonlocal conservation laws*
2019-06	**M. Colombo, G. Crippa, M. Graff and L. V. Spinolo** *On the role of numerical viscosity in the study of the local limit of nonlocal conservation laws*
2019-07	**G. Ciampa, G. Crippa and S. Spirito** *Smooth approximation is not a selection principle for the transport equation with rough vector field*
2019-08	**G. Ciampa, G. Crippa and S. Spirito** *Weak solutions obtained by the vortex method for the 2D Euler equations are Lagrangian and conserve the energy*
2019-09	**H. Harbrecht and I. Kalmykov** *Sparse grid approximation of the Riccati operator for closed loop parabolic control problems with Dirichlet boundary control*
2019-10	**J. Dölz, H. Harbrecht, S. Kurz, M. Multerer, S. Schöps, F. Wolf** *Bembel: The Fast Isogeometric Boundary Element C++ Library for Laplace, Helmholtz, and Electric Wave Equation*
2019-11	**Ch. Bürli, H. Harbrecht, P. Odermatt, S. Sayasone, N. Chitnis** *Age dependency in the transmission dynamics of the liver fluke, Opisthorchis viverrini and the effectiveness of interventions*
2019-12	**R. Baker and D. Masser** *Siegel's lemma is sharp for almost all linear systems*
2019-13	**H. Harbrecht, D. Tröndle and M. Zimmermann** *Approximating solution spaces as a product of polygons*