Optim Eng manuscript No.
(will be inserted by the editor)

Data-driven construction of Convex Region Surrogate models

**Qi Zhang · Ignacio E. Grossmann ·
Arul Sundaramoorthy · Jose M. Pinto**

Received: date / Accepted: date

Abstract As we strive to solve more complex and integrated optimization problems, there is an increasing demand for process models that are sufficiently accurate as well as computationally efficient. In this work, we develop an algorithm for the data-driven construction of a type of surrogate models that can be formulated as mixed-integer linear programs yet still provide good approximations of nonlinearities and nonconvexities. In such a surrogate model, which we refer to as Convex Region Surrogate, the feasible region is given by the union of convex regions in the form of polytopes and for each region, the objective function can be approximated by a linear function. In this paper, we present the proposed two-phase algorithm and demonstrate its effectiveness with a real-world case study.

Keywords Surrogate modeling · Data-driven modeling · Mixed-integer programming · Polyhedral theory · Multiscale optimization

1 Introduction

This paper deals with the following problem: Given a set of data points in the K-dimensional parameter space and scalar cost values associated with each data point, find a set of convex regions in the K-dimensional space such that the union of the convex regions describes a tight envelope around all data points, and for the data in each region, find a linear cost correlation within a prespecified error tolerance. This is a fairly generic problem and in our proposed algorithm, we apply concepts from

Q. Zhang · I.E. Grossmann
Center for Advanced Process Decision-making, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
E-mail: grossmann@cmu.com

A. Sundaramoorthy
Praxair Inc., Business and Supply Chain Optimization R&D, Tonawanda, NY 14150, USA

J.M. Pinto
Praxair Inc., Business and Supply Chain Optimization R&D, Danbury, CT 06810, USA

various areas, such as mixed-integer programming, polyhedral theory, metamodeling (Simpson et al 2001), clustering (Jain 2010), and pattern recognition (Jain et al 2000). However, to the best of our knowledge, the problem as such has not been reported in the literature before.

This work is motivated by the high demand for computationally efficient process models in many engineering applications. Typically, when constructing a mathematical model, we have to trade off model accuracy against computational tractability, especially if we want to use the model for optimization purposes. If the computational budget forbids the use of a detailed process model, we have to create a surrogate model that can be used instead. As the trend goes towards integrated optimization involving multiple levels of decision-making, e.g. in dynamic real-time optimization or planning and scheduling, there is an increasing demand for high-quality surrogate models that are computationally efficient and at the same time ensure a sufficient degree of accuracy (Chung et al 2011).

Surrogate modeling techniques can be divided into two general categories: model order reduction and data-driven modeling (Biegler et al 2014). In model order reduction, one starts with a high-fidelity model and tries to reduce the complexity of that model while retaining most of the structure of the original equations. Data-driven modeling, on the other hand, solely relies on data and does not make use of the explicit formulation of an existing model. Data-driven modeling approaches have become popular in recent years because of their wide applicability (Queipo et al 2005; Wang and Shan 2007) and are used in one of the following three situations: (1) The existing model is too complex to be reduced to the desired degree by using model order reduction techniques. (2) There exists a model that can be used for simulation, but we do not have access to the explicit model formulation. (3) A mathematical model of the process does not exist, yet we can draw data from the actual process.

A major challenge in surrogate modeling is the approximation of nonlinearities and nonconvexities, in which case most existing methods have to apply nonlinear structures in order to achieve good accuracy. However, in mixed-integer programming, there is typically a huge difference in computational complexity between a linear and a nonlinear formulation. Thus, we want to avoid nonlinearities when constructing surrogate models for mixed-integer programming frameworks, which frequently arise in various engineering applications, such as flowsheet design, planning and scheduling, and supply chain optimization.

Ierapetritou (2001) applies the concept of flexibility analysis (Swaney and Grossmann 1985) and uses the Quickhull algorithm (Barber et al 1996) to find a convex hull as approximation of the feasible region of a given model. The constructed convex hull is guaranteed to be inscribed in the actual feasible region, however, only if the feasible region is convex. To evaluate the feasibility of nonconvex processes, Goyal and Ierapetritou (2003) develop an approach in which the feasible region is approximated by subtracting outer polytopes around the infeasible regions from an expanded convex hull obtained by simplicial approximation (Goyal and Ierapetritou 2002). Sung and Maravelias (2007) propose an attainable region approach that determines the feasible production levels of the underlying MILP production scheduling model and an underestimation of the production cost in an integrated planning and scheduling framework. In a subsequent work (Sung and Maravelias 2009), the same authors develop an extension of the attainable region approach that takes nonconvexities into account by combining multiple polytopes.

The approaches reviewed above all require the full mathematical formulation of the model that we want to approximate. Despite the general applicability of data-driven methods, the number of related works taking an approach similar to ours is very limited. Karwan and Keblis (2007) use the convex hull around given data points as an approximation of the feasible region and apply linear regression to find a linear surrogate cost correlation. This method is easy to implement and has been successfully applied in other works such as Mitra et al (2012). However, the resulting surrogate model is only accurate if the true feasible region is convex and the true cost function is linear. Üney and Türkay (2006) propose an interesting application of mixed-integer programming in mulit-class data classification in which hyperboxes are used to define the regions for the different classes.

In this work, we propose to approximate the feasible region with the union of convex regions in the form polytopes. Furthermore, for each region, the objective function is approximated by a linear function. We show that such a surrogate model, which we refer to as Convex Region Surrogate (CRS), can be formulated as a mixed-integer linear program (MILP). Thus, CRS models are ideal for problems that are already formulated as MILPs since in that case, including a CRS model in the formulation of the optimization problem will not increase its structural complexity. The main contribution of this work is the development of an algorithm for the data-driven construction of CRS models. The proposed algorithm takes data drawn from the process (or simulations), and constructs convex regions such that the union of the convex regions describes a tight envelope around all data points and a sufficiently accurate linear approximation of the objective function can be found for each region.

This paper is organized as follows. In Section 2, we explain the concept of CRS models and their advantages before formally stating the problem of constructing CRS models in Section 3. Section 4 shows a brief illustrative overview of the proposed algorithm which is divided into two phases, Phase 1 and Phase 2. Accompanied by an illustrative example, the algorithms for Phase 1 and Phase 2 are described in detail in Section 5 and Section 6, respectively. After summarizing the complete algorithm in Section 7, we demonstrate its effectiveness by applying it to a real-world case study in Section 8. Finally, in Section 9, we close with a summary of this work and some concluding remarks.

2 Convex Region Surrogate

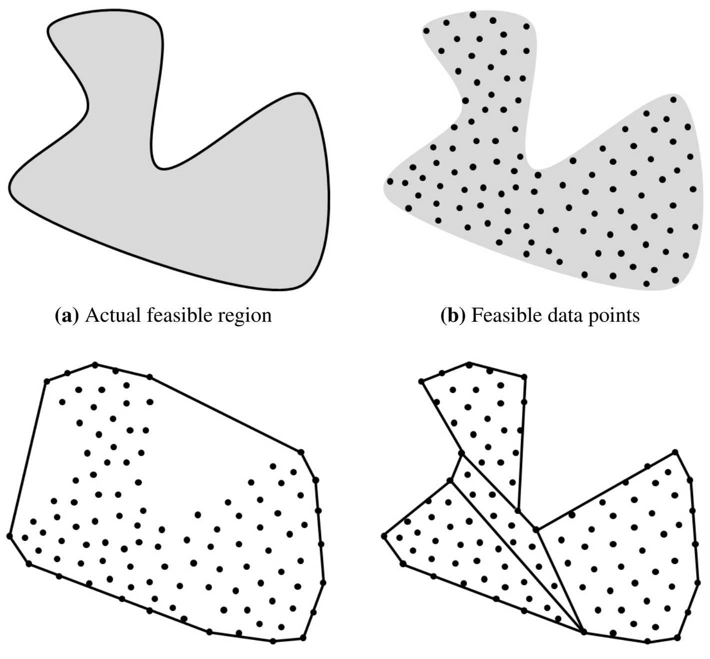
Optimization models have the general form

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in X \end{aligned} \tag{1}$$

where f is the scalar cost function, x is the vector of variables, and X describes the set of all feasible x. The goal of surrogate modeling is to construct an approximation of the feasible region X as well as the cost function $f(x)$.

The concept of the CRS is inspired by Karwan and Keblis (2007) who propose to approximate the feasible region with the convex hull around all given data points, and apply linear regression to the data to find a linear approximation of the cost correlation. The advantage of this approach is that the model remains linear and convex. Also, one can easily reduce the dimension of the feasible region by only considering relevant variables, which is especially useful in multiscale optimization. For example, in production scheduling, we often only need to know how much the plant can produce. Thus, in that case, instead of considering all variables including temperatures, pressures and intermediate flows, we would only include the production rates in our surrogate model.

However, the approach by Karwan and Keblis (2007) has two major limitations. First, if the true cost correlation is nonlinear, linear regression over all given data points will be inaccurate. Second, if the true feasible region is nonconvex, the convex hull around all data points will provide a poor approximation. Let us illustrate the second point with an example. Suppose we want to approximate the nonconvex feasible region shown in Fig. 1a and we are given the data points shown in Fig. 1b. If we just take the convex hull, we obtain an approximation of the feasible region as shown in Fig. 1c. Clearly, this is not a very good approximation because it contains significant amount of empty areas that do not belong to the actual feasible region but are considered feasible in the surrogate model.


(a) Actual feasible region **(b)** Feasible data points

(c) Convex hull around the data points **(d)** Union of four convex regions

Fig. 1: Data points are sampled from the feasible region. The nonconvex feasible region can be approximated more accurately by the union of multiple convex regions.

Instead of using the convex hull, we propose to approximate the feasible region with the union of convex regions in the form of polytopes. In this way, as illustrated in Fig. 1d, we can obtain a considerably more accurate representation of the feasible region. In addition, if the true cost function is not linear, we can construct the regions such that a sufficiently accurate linear approximation of the cost correlation can be found for the data of each region. In this way, we obtain a piecewise linear approximation of the cost function.

Since each region is a polytope, any point in a region can be expressed as a convex combination of its vertices. A feasible point has to be in one of the regions. Furthermore, the form of the cost function depends on the region in which the feasible point lies. As a result, the CRS model can be naturally described with a disjunction (Grossmann and Trespalacios 2013) and formulated as the following generalized disjunctive program (GDP):

$$\begin{aligned} \min \quad & f \\ \text{s.t.} \quad & \bigvee_{r \in R} \begin{bmatrix} Y_r \\ x = \sum_{j \in V_r} \lambda_j v_{rj} \\ \sum_{j \in V_r} \lambda_j = 1 \\ 0 \leq \lambda_j \leq 1 \;\; \forall j \in V_r \\ f = b_r + c_r^{\mathrm{T}} x \end{bmatrix} \\ & \underline{\bigvee}_{r \in R} Y_r \\ & Y_r \in \{true, false\} \;\; \forall r \in R \end{aligned} \tag{2}$$

where R is the set of convex regions. In each region r, x is described as the convex combination of vertices v_{rj} with $j \in V_r$, that correspond to the set of vertices in region r. The nonnegative multipliers, which have to sum up to 1 over $j \in V_r$, are denoted by λ_j. b_r and c_r are the cost constant and coefficients for the cost correlation in region r. Y_r is a boolean variable and takes the value *true* if the chosen feasible point lies in region r.

By applying the hull reformulation (Grossmann and Trespalacios 2013), the GDP can be transformed into the following MILP:

$$\begin{aligned} \min \quad & \sum_{r \in R} \left(b_r y_r + c_r^{\mathrm{T}} x \right) \\ \text{s.t.} \quad & x = \sum_{r \in R} \bar{x}_r \\ & \bar{x}_r = \sum_{j \in V_r} \bar{\lambda}_{rj} v_{rj} && \forall r \in R \\ & \sum_{j \in V_r} \bar{\lambda}_{rj} = y_r && \forall r \in R \\ & 0 \leq \bar{\lambda}_{rj} \leq 1 && \forall r \in R, j \in V_r \\ & \sum_{r \in R} y_r = 1 \\ & y_r \in \{0, 1\} && \forall r \in R \end{aligned} \tag{3}$$

where x and λ_j are disaggregated into the region-dependent variables $\bar{x}_r$ and $\bar{\lambda}_{rj}$, respectively. $\bar{x}_r$ and $\bar{\lambda}_{rj}$ can only be nonzero if the region r is chosen. The binary variables y_r have to sum up to 1 over $r \in R$, i.e. only one region can be chosen. In the cost function, the constants and coefficients also only apply if x lies in the corresponding region.

The main advantages of a CRS model are the following:

- Can approximate nonlinear and discontinuous cost functions.
- Can approximate nonlinear and nonconvex feasible regions.
- By formulating the CRS model as an MILP, we can make use of efficient MILP solvers such as CPLEX and GUROBI. This is especially useful if the optimization problem, in which we want to integrate the CRS model, is already formulated as an MILP.
- The dimension of the variable space in a CRS model can be minimized by only considering data in the desired space.

In order to formulate a CRS model, we need to know the set of regions R, the corresponding sets of vertices V_r, as well as the region-dependent cost constants and coefficients. Finding these components is a nontrivial task for which we present an algorithm in the remainder of this paper.

3 Formal problem statement

After we demonstrated in the previous section how to formulate a CRS model as an MILP given a disjunctive set of convex regions, we take on the task of constructing suitable convex regions and their cost approximations from data. This problem is formally stated in the following.

Given are n data points where each data point j consists of a K-dimensional parameter vector $a_j \in \mathrm{R}^K$ and a scalar cost value g_j. Find convex regions r in the form of polytopes in the parameter space such that

- the union of the convex regions contains all a_j,
- no a_j lies in the interior of two or more convex regions, i.e. the convex regions do not overlap except for possibly at the boundaries,
- in each region r, there exists a linear correlation (with a maximum error ε) between the parameter and the cost values of the data points contained in region r, i.e.

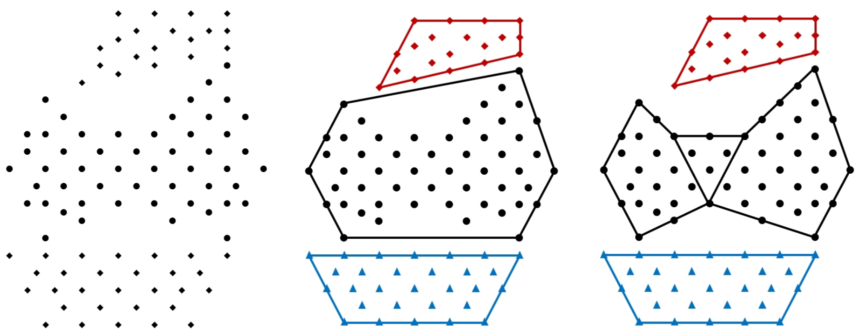
$$\bar{g}_j = b_r + c_r^{\mathrm{T}} a_j = b_r + \sum_{k=1}^{K} c_{rk} a_{jk}\,, \quad |\,\bar{g}_j - g_j\,| \leq \varepsilon \quad \forall j \in J_r \tag{4}$$

  where J_r is the set of data points contained in region r,
- the union of the convex regions represents a tight envelope for the feasible region indicated by the given data points.

4 Illustrative overview of algorithm

To obtain the Convex Region Surrogate (CRS), we propose a two-phase algorithm. In Phase 1, the set of all data points is divided into subsets such that a linear parameter-cost correlation can be obtained within a tolerance for all points in each subset, and that the convex hulls constructed around the points of each subset do not overlap. In Phase 2, for each subset, the feasible region indicated by the data points assigned to the subset is approximated by constructing multiple convex regions of which the union contains all points of the subset.

In the next two sections, we present the Phase 1 and Phase 2 algorithms in detail. In order to facilitate the understanding of the algorithm, the explanation of each step is accompanied by an illustrative two-dimensional example. The data points for this example are shown in Fig. 2a, in which the different markers indicate different parameter-cost correlations. The complete set of data for this illustrative example can be found in the appendix section.



(a) Given data points **(b)** At the end of Phase 1 **(c)** At the end of Phase 2

Fig. 2: (a) Points marked by circles (center area) have the same linear cost correlation. The remaining points, marked by diamonds, have a different linear cost correlation. (b)(c) Assignment of data points and resulting polytopes from Phase 1 and 2.

Fig. 2b and 2c show the results at the end of Phase 1 and Phase 2. In Phase 1, the data points are assigned into three disjoint subsets. Note that we obtain three subsets although there are only two different parameter-cost correlations because with two subsets, the resulting two convex hulls would overlap. In Phase 2, we check for each subset whether it needs to be further partitioned such that multiple convex regions can be obtained to construct a tighter envelope of the feasible region. Here, this is only necessary for the second subset, for which three convex regions are constructed to approximate the feasible region (c.f. Fig. 2c).

5 Phase 1: Subset assignment

In Phase 1, we assign the given data points to subsets such that linear parameter-cost correlations can be obtained for each subset and that the convex hulls of the subsets do not overlap. A flowchart for the Phase 1 algorithm is shown in Fig. 3. First, we set the initial number of subsets m, typically to 1. Now we try to assign the data points to m subsets such that linear parameter-cost correlations can be obtained within the tolerance for each subset. We increase m until the assignment problem is feasible. In the next step, the vertices of the convex hulls for each subset are obtained. Using these, we can then check whether the convex hulls overlap each other. If they do, we add cuts to the subset assignment problem in order to avoid obtaining the same solution in the next iteration. The algorithm terminates when the assignment solution provides m subsets such that the corresponding convex hulls are disjoint.

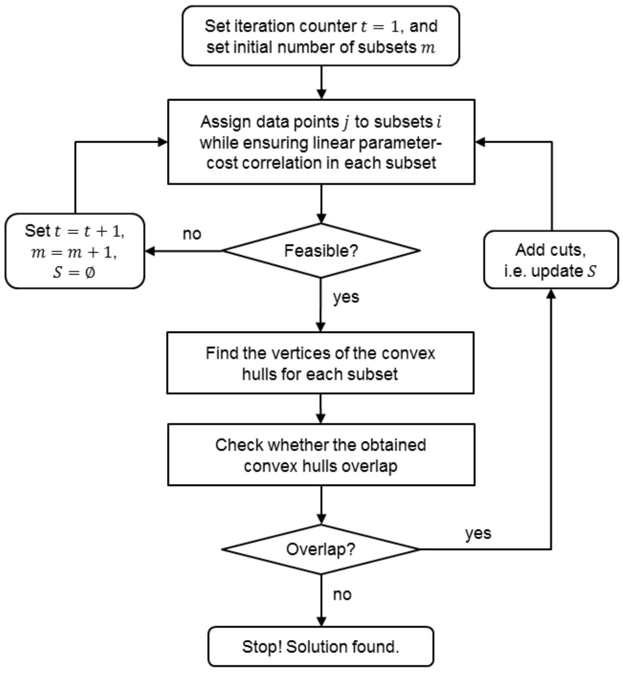

Set iteration counter $t = 1$, and set initial number of subsets m
Assign data points j to subsets i while ensuring linear parameter-cost correlation in each subset
Set $t = t + 1$, $m = m + 1$, $S = \emptyset$
no
Feasible?
Add cuts, i.e. update S
yes
Find the vertices of the convex hulls for each subset
Check whether the obtained convex hulls overlap
yes
Overlap?
no
Stop! Solution found.

Fig. 3: Flowchart for the Phase 1 algorithm

5.1 Subset assignment formulation

At the heart of the Phase 1 algorithm is the assignment problem, which assigns n data points to m subsets. For the set of subsets at iteration t, $I^t = \{1, 2, \ldots, m\}$, we find a feasible solution to the following set of mixed-integer linear constraints:

$$\sum_{i \in I^t} y_{ij} = 1 \qquad \forall j \tag{5a}$$

$$\sum_{j} y_{ij} \geq 1 \qquad \forall i \in I^t \tag{5b}$$

$$\bar{g}_{ij} = b_i + \sum_{k=1}^{K} c_{ik} a_{jk} \qquad \forall i \in I^t, j \tag{5c}$$

$$\varepsilon_{ij} = \frac{\bar{g}_{ij}}{g_j} - 1 \qquad \forall i \in I^t, j \tag{5d}$$

$$-\varepsilon - M(1 - y_{ij}) \le \varepsilon_{ij} \le \varepsilon + M(1 - y_{ij}) \qquad \forall i \in I^t, j \tag{5e}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i \in I^t, j \tag{5f}$$

where the binary variable $y_{ij} = 1$ if point j is assigned to subset i. Constraints (5a) and (5b) ensure that every point is assigned to a subset, and that every subset contains at least one point. Eq. (5c) describes the linear correlation for each subset i applied to all points j, from which the fitting error ε_{ij} is calculated in Eq. (5d). ε is a prespecified error tolerance, and constraint (5e) forces ε_{ij} to be bounded by $-\varepsilon$ and ε if point j is assigned to subset i. Otherwise, the constraint is relaxed.

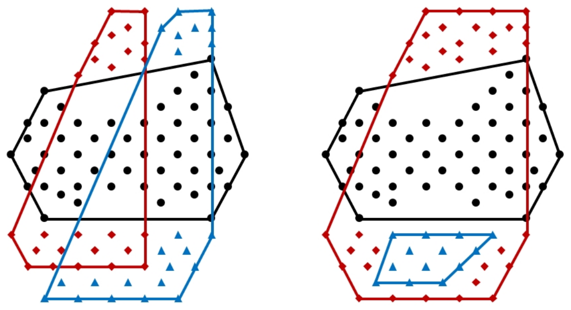


Fig. 4: For the illustrative example, the constraint set given by (5) with $m = 3$ has many solutions which lead to overlapping convex hulls.

(5) is only a feasibility problem and often, the solution is not unique, which may lead to the algorithm requiring a large number of iterations if many of the possible solutions do not result in non-overlapping convex hulls. For our illustrative example, Fig. 4 shows two possible solutions for $m = 3$ in which the resulting convex hulls overlap. Notice that such solution would be discouraged if we minimized the distances between the points in the same subset. To achieve this clustering effect, we propose the following alternative mixed-integer linear programming (MILP) formulation:

$$\min \quad \sum_{i \in I^t} \sum_{k} d_{ik} \tag{6a}$$

$$\text{s.t.} \quad \text{set of equations (5)} \tag{6b}$$

$$y_{ij} \ge z_{ijj'} \qquad \forall i \in I^t, j, j' > j \tag{6c}$$

$$y_{ij'} \ge z_{ijj'} \qquad \forall i \in I^t, j, j' > j \tag{6d}$$

$$z_{ijj'} \ge y_{ij} + y_{ij'} - 1 \qquad \forall i \in I^t, j, j' > j \tag{6e}$$

$$-d_{ik} \le z_{ijj'} \left(a_{jk} - a_{j'k}\right) \le d_{ik} \qquad \forall i \in I^t, j, j' > j, k \tag{6f}$$

$$y_{ij} \in \{0,1\} \qquad \forall\, i \in I^t,\, j \tag{6g}$$

$$z_{ijj'} \in [0,1] \qquad \forall\, i \in I^t,\, j,\, j' > j \tag{6h}$$

where d_{ik} denotes the maximum distance in the k-th dimension between two points in subset i, and $z_{ijj'} = 1$ if both points j and j' are assigned to subset i. Constraints (6c) – (6e) ensure that $z_{ijj'} = 1$ if and only if $y_{ij} = y_{ij'} = 1$. The maximum distances between two points in each subset are determined by including constraint (6f) and minimizing $\sum_{i \in I^t} \sum_k d_{ik}$. This formulation facilitates – but does not guarantee – subset assignment that results in non-overlapping convex hulls. In fact, in the case of the illustrative example, it provides the desired solution shown in Fig. 2b in one iteration. However, (6) is a significantly larger problem than (5). Therefore, with a large number of data points, it may be computationally more efficient to use (5) or both formulations in combination.

5.2 Constructing convex hulls

After obtaining a feasible solution to the subset assignment problem, we apply the Quickhull algorithm (Barber et al 1996) to find the vertices of the corresponding convex hulls. Alternatively, we can solve the following MILP for each subset $i \in I^t$ in order to find the respective vertices.

$$\min \quad \sum_{j \in J_i} x_j \tag{7a}$$

$$\text{s.t.} \quad a_j = \sum_{j' \in J_i} \lambda_{j'j}\, a'_j \qquad \forall\, j \in J_i \tag{7b}$$

$$\lambda_{jj'} \le x_j \qquad \forall\, j,\, j' \in J_i \tag{7c}$$

$$\sum_{j' \in J_i} \lambda_{j'j} = 1 \qquad \forall\, j \in J_i \tag{7d}$$

$$\lambda_{jj'} \ge 0 \qquad \forall\, j,\, j' \in J_i \tag{7e}$$

$$x_j \in \{0,1\} \qquad \forall\, j \in J_i \tag{7f}$$

where the binary variable $x_j = 1$ if point j is used in a convex combination. Since we are minimizing $\sum_j x_j$, the resulting objective function value will be the minimum number of points required to express all points in the subset as convex combination of the chosen points. Thus, the set of vertices for subset i is then $V_i = \left\{ j \in J_i : x_j = 1 \right\}$.

5.3 Identifying overlapping convex hulls

After having constructed the convex hulls, we have to check if they overlap. If two convex hulls overlap, we can find a point that belongs to both convex hulls. Otherwise, we cannot. We can determine the existence of such a point for each pair of convex hulls i and i' by checking the feasibility of the following set of equations:

$$p = \sum_{j \in V_i} \lambda_j\, a_j \tag{8a}$$

$$p = \sum_{j \in V_{i'}} \mu_j a_j \tag{8b}$$

$$\sum_{j \in V_i} \lambda_j = \sum_{j \in V_{i'}} \mu_j = 1 \tag{8c}$$

$$\lambda_j \geq 0 \qquad \forall j \in V_i \tag{8d}$$

$$\mu_j \geq 0 \qquad \forall j \in V_{i'} \tag{8e}$$

where V_i and $V_{i'}$ are the sets of vertices of convex hulls i and i', respectively. Clearly, point p is constrained to be inside both convex hulls i and i'. Thus, (8) is only feasible if such a point exists, i.e. if convex hulls i and i' overlap.

Instead of solving (8) for each pair of convex hulls, we detect all overlapping convex hulls by solving one single linear program (LP):

$$\min \quad \sum_{i \in I^t} \sum_{i' \in I^t, i < i'} \sum_k \left(s^+_{ii'k} + s^-_{ii'k}\right) \tag{9a}$$

$$\text{s.t.} \quad p_{ii'k} = \sum_{j \in V_i} \lambda_{ii'j} a_{jk} \qquad \forall i, i' \in I^t, i < i', k \tag{9b}$$

$$p_{ii'k} = \sum_{j \in V_{i'}} \mu_{ii'j} a_{jk} + s^+_{ii'k} - s^-_{ii',k} \qquad \forall i, i' \in I^t, i < i', k \tag{9c}$$

$$\sum_{j \in V_i} \lambda_{ii'j} = \sum_{j \in V_{i'}} \mu_{ii'j} = 1 \qquad \forall i, i' \in I^t, i < i' \tag{9d}$$

$$\lambda_{ii'j} \geq 0 \qquad \forall i, i' \in I^t, i < i', j \in V_i \tag{9e}$$

$$\mu_{ii'j} \geq 0 \qquad \forall i, i' \in I^t, i < i', j \in V_{i'} \tag{9f}$$

$$s^+_{ii'k}, s^-_{ii'k} \geq 0 \qquad \forall i, i' \in I^t, i < i', k \tag{9g}$$

Here we introduce slack variables $s^+_{ii'k}$ and $s^-_{ii'k}$ for each pair of convex hulls i and i' and each dimension k. Similarly as point p in (8), point $p_{ii'}$ is constrained to be inside convex hull i. However, according to Eq. (9c), point $p_{ii'}$ is only also inside convex hull i' if the corresponding slack variables are zero. Since LP (9) minimizes the sum of all slack variables, all slack variables corresponding to pairs of convex hulls for which a common point can be found will become zero at the optimal solution. In other words, we know that convex hulls i and i' overlap if the LP solution yields $\sum_k \left(s^+_{ii'k} + s^-_{ii'k}\right) = 0$. If they do not overlap, the LP solution yields $\sum_k \left(s^+_{ii'k} + s^-_{ii'k}\right) > 0$ since the point $p_{ii'}$ cannot be obtained as a convex combination of vertices of the convex hulls i and i'.

In the illustrative example, we encounter this situation the first time at $m = 2$. With the number of subsets set to 2, there is only one feasible solution to the subset assignment problem (5). As shown in Fig. 5, the two resulting convex hulls overlap, which can be detected by solving (9). As a result, the algorithm increases m to 3.

5.4 Overlap elimination cuts

The cuts are constraints that are added to the subset assignment problem (5) in order to eliminate previous solutions which have led to overlapping convex hulls. In fact,

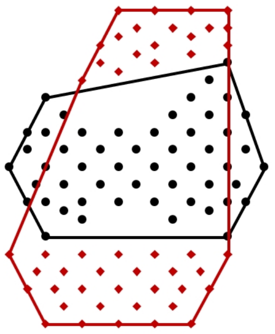
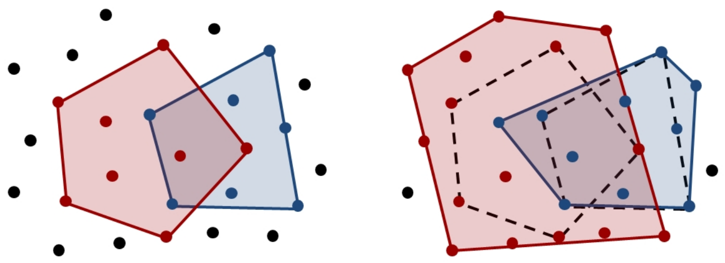


Fig. 5: For $m = 2$, the only feasible solution leads to two overlapping convex hulls.

we add cuts that cut off more than just one assignment solution at each iteration. The idea is that if we know from solving (9) that convex hulls i and i' overlap, we construct cuts that eliminate all assignment solutions which involve two subsets such that one of the two subsets contains the vertices of convex hull i and the other subset contains the vertices of convex hull i'.

We illustrate the idea of the proposed cuts with an example, shown in Fig. 6. Suppose we detected the two overlapping convex hulls, say i and i', shown in Fig. 6a. In any assignment solution, in which the vertices of convex hull i are assigned to one subset and the vertices of convex hull i' are assigned to another subset, the convex hulls resulting from these subsets will contain convex hulls i and i' and will therefore also overlap each other. This is true regardless which other points are assigned to the subsets. An example of such an assignment is shown in Fig. 6b. As depicted in the figure, the convex hulls i and i' (dashed lines) are inscribed in the new convex hulls.



(a) Pair of overlapping convex hulls, cuts will eliminate this assignment solution

(b) Cuts will also eliminate, for example, this assignment solution.

Fig. 6: The added cuts eliminate assignment solutions guaranteed to result in overlapping convex hulls based on information from the current solution.

Now let us formulate the cuts mathematically. The cuts are accumulated in the set S, which is set to $\emptyset$ when the number of subsets m is increased, i.e. S is empty at the beginning of each iteration t. For each cut $s \in S$, we define a set $\bar{I}_s = \{i, i'\}$ with i and

i' being two convex hulls overlapping each other. The cuts in terms of the assignment variables y_{ij} for a given point j in subset i are formulated as follows:

$$\sum_{i \in \bar{I}_s} \sum_{j \in \bar{V}_{is}} y_{ij} \leq \sum_{i \in \bar{I}_s} |\bar{V}_{is}| - 1 \qquad \forall s \in S,\, i \in \bar{I}_s, \tag{10}$$

where $\bar{V}_{is}$ is the set of vertices of convex hull i for cut s and $|\bar{V}_{is}|$ is the cardinality of set $\bar{V}_{is}$. In the Integer Programming literature, (10) is generally known as cover cut (Crowder et al 1983). Note that for each pair of overlapping convex hulls, we have to add $\binom{m}{2}$ cuts in order to account for all possible combinations of subset indices. For example, if we have 3 subsets, we have to add $\binom{3}{2} = 3$ cuts for each pair of overlapping convex hulls, i.e. construct one cut for each of the three subset index pairs $\{1,2\}$, $\{1,3\}$, and $\{2,3\}$.

6 Phase 2: Construction of convex regions

In Phase 2, we construct convex regions for each subset i such that the union of the convex regions results in an accurate approximation of the generally nonconvex feasible region. In our illustrative example, the feasible regions for the first and third subsets are convex such that the convex hulls resulting from Phase 1 are already the tightest approximation (c.f. Fig. 2b). Therefore, we will demonstrate the Phase 2 algorithm with the second subset of which the feasible region is clearly nonconvex.

Phase 2 consists of two main steps, as illustrated in Fig. 7. First, we detect the contour of the feasible region, i.e. we find its vertices and facets. In the second step, the resulting envelope for the feasible region is partitioned into polytopes such that the union of these polytopes forms the desired surrogate feasible region.

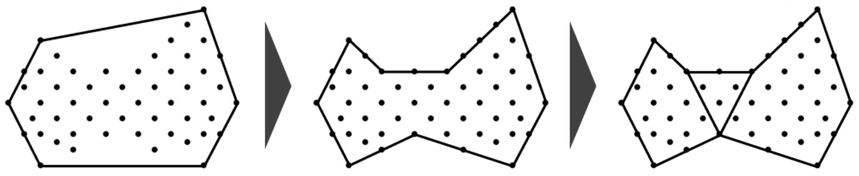
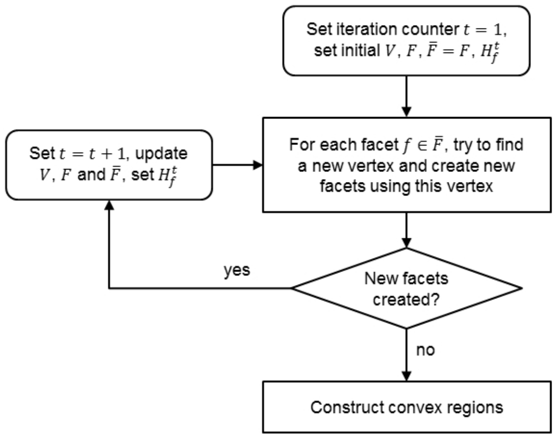


Fig. 7: Starting from the convex hull of the given data points, the Phase 2 algorithm first finds the contour of the tight envelope around the feasible region and subsequently constructs the polytopes representing the envelope.

In the following, we will first give an overview of the Phase 2 algorithm and then describe each part in detail. Note that since Phase 2 can be performed on each subset independently, we omit the index i in the following formulations for the sake of readability.

The high-level flowchart for the Phase 2 algorithm is shown in Fig. 8. The algorithm is initialized with results obtained in Phase 1, namely the set of vertices V and the set of facets F of the convex hull. The initial $\bar{F}$, which is the set of facets to be examined in the current iteration, is set to F. Furthermore, for each facet f, we define the set H_f which contains the vertices of facet f.



Set iteration counter $t = 1$, set initial V, F, $\bar{F} = F$, H_f^t
For each facet $f \in \bar{F}$, try to find a new vertex and create new facets using this vertex
Set $t = t + 1$, update V, F and $\bar{F}$, set H_f^t
New facets created?
yes
no
Construct convex regions

Fig. 8: Overview of the Phase 2 algorithm

As illustrated in Fig. 7, the contour of the feasible region can be approximated by facets which are defined by the corresponding vertices. At each iteration, we move into the current outer approximation of the feasible region as far as possible in order to find new vertices and construct new facets using those new vertices. The idea is to take each facet $f \in \bar{F}$ and try to find a data point with which new facets can be formed without cutting off any data points. The point, which fulfills this condition and is sufficiently far away from the facet, is then declared a new vertex and new facets are formed using this vertex. In this way, a tighter outer approximation of the feasible region is formed. At each iteration, if new facets have been created, the sets V, F, $\bar{F}$, and H_f^t are updated before the algorithm moves to the next iteration. This process is repeated until no more new vertices and facets can be found. Note that after new vertices are added to the initial set V, the set of vertices will not define a convex polytope anymore but rather a nonconvex polygon. The algorithm for finding new vertices and creating new facets is described in detail in Section 6.1.

After the contour of the feasible region is obtained, the algorithm uses the information about the vertices and facets to construct the desired convex regions. At the heart of this process is an assignment problem, which assigns facets, vertices and data points to convex regions. We will elaborate on this part of Phase 2 in Section 6.2.

6.1 Detecting the contour of the feasible region

At each iteration, for each facet $f \in \bar{F}$, the algorithm shown in Fig. 9 is applied to find one new vertex and create new facets by connecting the new vertex with vertices of facet f. We use two criteria to identify the new vertex. First, to obtain an outer approximation of the feasible region, the new facets created using the new vertex must not cut off any data points. Second, to move as far as possible into the current approximation of the feasible region, we look for the data point that is furthest away from facet f while satisfying the first condition.

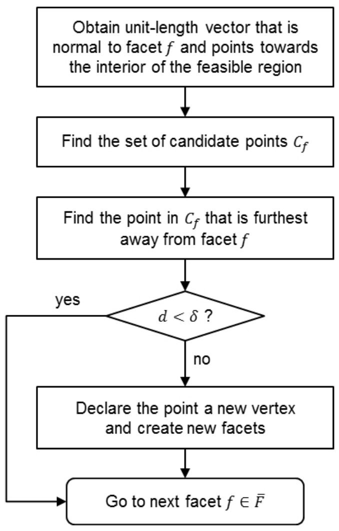

Obtain unit-length vector that is normal to facet f and points towards the interior of the feasible region
Find the set of candidate points C_f
Find the point in C_f that is furthest away from facet f
yes
$d < \delta$?
no
Declare the point a new vertex and create new facets
Go to next facet $f \in \bar{F}$

Fig. 9: Flowchart of the algorithm applied to each facet $f \in \bar{F}$ to find new vertices and create new facets

In the proposed algorithm, we first obtain a unit-length vector that is perpendicular to the facet and points towards the interior of the feasible region. We then find the set of all candidate points C_f where a candidate point is a data point which fulfills the first condition stated above. The normal vector is used in the next step to measure the distance to the facet so that we can determine the point in C_f that is furthest away from the facet. If the distance d is greater than or equal a specified tolerance δ, we declare this point a new vertex, create the corresponding new facets and then move on to the next facet. Otherwise, we directly go to the next facet $f \in \bar{F}$.

Remark 1 Often in practice, one wants to avoid creating new facets from facets that are already very small in size. Therefore, in our implementation, we determine the maximum Euclidean distance between two vertices of a facet and only further examine the facet if the distance is greater than a prespecified value.

6.1.1 Obtaining normal vectors

To find a normal vector for facet f, we first obtain an expression for the hyperplane which contains the facet. For this, we need K points that are on the hyperplane, for which we simply take the first K vertices of the facet. Suppose these points are

$a_1, a_2, \ldots, a_K$, then any point p on the hyperplane can be expressed as

$$p = b_f^0 + \sum_{k'=1}^{K-1} \alpha_{k'} b_{fk'} \tag{11}$$

where $b_f^0 = a_1$, $b_{fk'} = a_{k'+1} - a_1$ are the difference vectors. Starting at point b_f^0, any point on the facet-containing hyperplane can be reached by varying the coefficients $\alpha_{k'}$ since the hyperplane is spanned by $b_{fk'}$. Note that $\alpha_{k'}$ are unrestricted in sign and magnitude.

To obtain a vector that is perpendicular to the facet and has unit length, we simply need to find an $n_f \in \mathrm{R}^K$ that is normal to all difference vectors, i.e. $b_f^{\mathrm{T}} n_f = 0$, and satisfies $\|n_f\|_2 = 1$. However, we need the normal vector to point towards the interior of the feasible region. Since the solution for n_f is not unique, we need to check in which direction n_f points and change the sign if it points outwards the feasible region.

The procedure for determining the direction of n_f is different for $t = 1$ and $t > 1$. At the first iteration ($t = 1$), the initial facets describe the convex hull around all data points of the subset. Hence, the facet-containing hyperplanes are supporting hyperplanes for the convex hull, i.e. all data points lie on one side of the hyperplane, as illustrated in Fig. 10. This implies that if n_f points towards the interior of the feasible region, we will have a positive directed distance d between the hyperplane and any given data point $\bar{a}$ that does not lie on the facet. We choose such a point $\bar{a}$ and calculate d by solving the following set of equations:

$$\begin{aligned} p &= b_f^0 + \sum_{k'=1}^{K-1} \alpha_{k'} b_{fk'} \\ \bar{a} &= p + d\, n_f \end{aligned} \tag{12}$$

where the variable p is a point on the hyperplane containing facet f. If the d resulting from (12) is positive, n_f points towards the interior of the feasible region and we set $\bar{n}_f = n_f$. If d is negative, n_f points outwards and we set $\bar{n}_f = -n_f$.

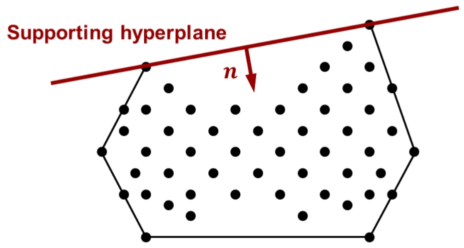


Fig. 10: All data points lie on one side of a supporting hyperplane.

The procedure described above only applies in the first iteration because the facet-containing hyperplanes in later iterations are not necessarily supporting hyperplanes for the whole feasible region anymore. However, for $t > 1$, we only have to consider the newly created facets and we can exploit the fact that we know for each new facet from which "old" facet it originates. As illustrated in Fig. 11, the entire old facet f' lies on one side of the hyperplane containing the corresponding new facet f. The normal vector of the new facet is supposed to point towards the other side. In order to check the direction in which n_f points, we formulate the following LP:

$$\max \quad d \tag{13a}$$

$$\text{s.t.} \quad p = \sum_{j \in H_f^t} \lambda_j a_j \tag{13b}$$

$$\sum_{j \in H_{f'}^{t-1}} \mu_j a_j = p + d n_f \tag{13c}$$

$$\sum_{j \in H_f^t} \lambda_j = \sum_{j \in H_{f'}^{t-1}} \mu_j = 1 \tag{13d}$$

$$\lambda_j \geq 0 \quad \forall j \in H_f^t \tag{13e}$$

$$\mu_j \geq 0 \quad \forall j \in H_{f'}^{t-1} \tag{13f}$$

where f' is the old facet from which facet f originates and $H_{f'}^{t-1}$ is the set of vertices of facet f'. As one can see from Eq. (13b), p is described as a convex combination of facet f's vertices and is therefore a point on facet f. We now reach a point on facet f' by moving from point p in the direction of n_f, as given by Eq. (13c). If n_f points in the direction of facet f', d will be positive at the optimal solution, in which case we set $\bar{n}_f = -n_f$. If n_f points into the feasible region, d will be zero at the optimum since the maximum d will be reached at the vertex shared by facet f and f' (point A in the example in Fig. 11). In this case, we set $\bar{n}_f = n_f$.

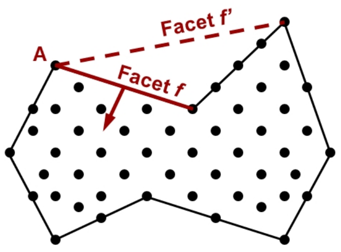


Fig. 11: The dashed line indicates the old facet which led to the two new facets.

6.1.2 Detecting points in the interior of a facet

For each facet, we want to find a data point which can be used in conjunction with the facet to form a polytope that can be cut off from the current approximation of the feasible region. This point will then be a new vertex describing the contour of the feasible region.

However, first we notice that we can only induce a polytope to be cut off from a facet that does not contain any data points in its interior. This is illustrated in the example shown in Fig. 12, in which we examine the top facet of the rectangular convex hull around all data points. In Fig. 12a, the facet's interior is empty, which in our 2-dimensional example simply means that there is no point located on the line between the vertices. Therefore, with this facet and another suitable data point, we can construct a polytope that can be cut off, as depicted by the shaded triangle. However, this would not be feasible if the facet had a point in its interior since then we would cut off that point, as illustrated in Fig. 12b.

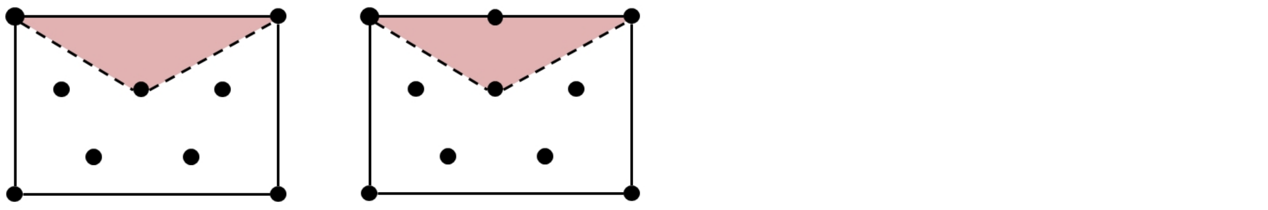

(a) No point in the facet's interior **(b)** Point in the facet's interior

Fig. 12: A facet that contains a point in its interior cannot be used to form a polytope that can be cut off.

We solve a simple LP for each facet f to determine if the facet's interior is empty. If it is not, we do not need to further examine that facet. With this LP, we check for each data point if it is in the interior of facet f. Since we know that this cannot be true for any of the vertices, we exclude the points in V from this examination. In the LP formulation, we express every a_j for $j \notin V$ as a convex combination of the vertices of facet f. We use slack variables to relax this constraint if a_j is not located on facet f and therefore cannot be expressed as a convex combination of the facet's vertices. The multipliers are further constrained to be greater than or equal to a small $\bar{\varepsilon}$ so that a_j will be a point in the interior of facet f if the corresponding slack variables are zero. The LP is formulated as follows:

$$\begin{aligned}
\min \quad & \sum_{j \notin V} \sum_{k} \left(s^+_{jk} + s^-_{jk} \right) && && \text{(14a)}\\
\text{s.t.} \quad & a_{jk} = \sum_{j' \in H^t_f} \lambda_{j'j} a_{j'k} + s^+_{jk} - s^-_{jk} && \forall j \notin V, k && \text{(14b)}\\
& \sum_{j' \in H^t_f} \lambda_{j'j} = 1 && \forall j \notin V && \text{(14c)}\\
& \lambda_{jj'} \geq \bar{\varepsilon} && \forall j \in H^t_f, j' \notin V && \text{(14d)}\\
& s^+_{jk}, s^-_{jk} \geq 0 && \forall j \notin V, k && \text{(14e)}
\end{aligned}$$

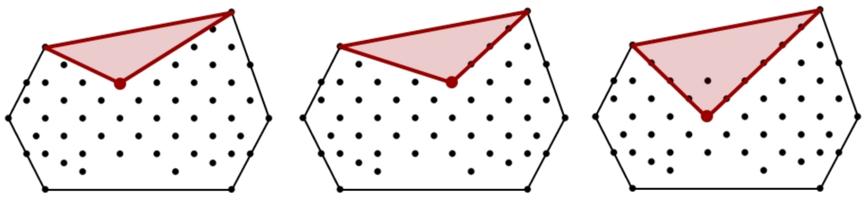
By solving (14), the sum of all slack variables is minimized. As a result, we will know that point j is located in the interior of facet f if $\sum_k \left(s^+_{jk} + s^-_{jk} \right) = 0$ at the optimum.

6.1.3 Identifying candidate points

For each facet f that does not contain any data points in its interior, we identify candidate points that can potentially become a new vertex. A candidate point for facet f is defined as follows.

Definition 1 Consider a data point j that is not a vertex. Form a polytope with point j as well as the vertices of facet f being vertices of this polytope, which we denote PT_{fj}. Point j is a candidate point for facet f if polytope PT_{fj} is completely contained in the current approximation of the feasible region and does not contain any data points in its interior.

In Fig. 13, the definition of a candidate point is illustrated for the top facet of our example. The polytope formed in a 2-dimensional case is always a triangle. In Fig. 13a, the triangle formed by the facet and the chosen data point does not contain any points except for the vertices. The chosen data point is therefore a candidate point. In Fig. 13b, the triangle contains additional data points, but only on the boundary. The corresponding chosen point here is therefore also a candidate point. In contrast, the chosen point in Fig. 13c is not a candidate point because the resulting triangle contains a data point in its interior.



(a) Candidate point **(b)** Candidate point **(c)** Not a candidate point

Fig. 13: A candidate point is found if the polytope formed with the facet does not contain any data points in its interior.

For each facet, we have to check for every data point j that is not a vertex, i.e. $j \notin V$, if this point is a candidate point. For this, we first solve the LP given in (15) for each point $j \notin V$. The LP could be feasible or infeasible. It is feasible if point j is on the correct side of the facet, i.e. it can be reached from the facet by moving in the direction of $\bar{n}_f$. Otherwise, the LP is infeasible. Furthermore, when feasible, the optimal solution of the LP will indicate if there are any data points lying in the interior of polytope PT_{fj}.

$$\min \quad \sum_{j' \notin H^t_f, j' \neq j} \sum_k \left(s^+_{j'k} + s^-_{j'k} \right) \tag{15a}$$

$$\text{s.t.} \quad p_k = \sum_{j' \in H^t_f} v_{j'} a_{j'k} \qquad \forall k \tag{15b}$$

$$\sum_{j' \in H_f^t} v_{j'} = 1 \tag{15c}$$

$$v_{j'} \geq 0 \qquad \forall j' \in H_f^t \tag{15d}$$

$$a_{jk} = p_k + d\,\bar{n}_{fk} \qquad \forall k \tag{15e}$$

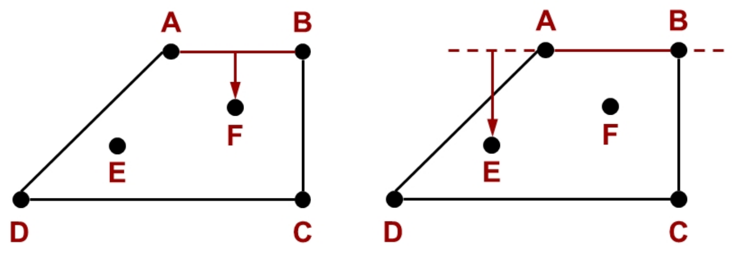
$$a_{j'k} = \lambda_{jj'}\, a_{jk} + \sum_{j'' \in H_f^t} \lambda_{j''j'}\, a_{j''k} + s^+_{j'k} - s^-_{j'k} \qquad \forall j' \notin H_f^t,\ j' \neq j,\ k \tag{15f}$$

$$\lambda_{jj'} + \sum_{j'' \in H_f^t} \lambda_{j'',j'} = 1 \qquad \forall j' \notin H_f^t,\ j' \neq j \tag{15g}$$

$$\lambda_{j''j'} \geq \bar{\varepsilon} \qquad \forall j' \notin H_f^t,\ j' \neq j,\ j'' = j \text{ or } \in H_f^t \tag{15h}$$

$$s^+_{j'k},\ s^-_{j'k} \geq 0 \qquad \forall j' \notin H_f^t,\ j' \neq j,\ k \tag{15i}$$

We first focus on Eqs. (15b) – (15e) before explaining the remaining constraints. Eqs. (15b) – (15e) ensure that the point j, that is being checked, is on the side of the facet towards which the normal vector $\bar{n}_f$ points. Actually, this formulation is slightly more restrictive as it states that point j has to be reached by moving from a point p on the facet in the direction of $\bar{n}_f$. This is illustrated in Fig. 14a, in which point F is the only non-vertex point that can be reached from facet A-B. Thus, LP (15) is feasible for point F but not for point E.



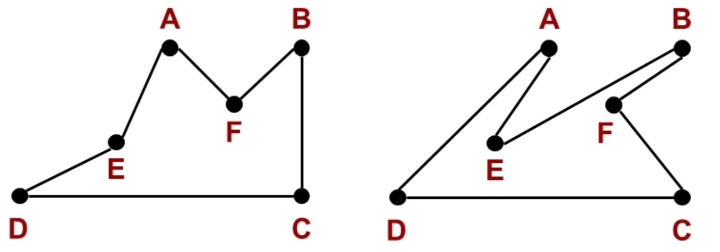
(a) From facet **(b)** From facet-containing hyperplane

Fig. 14: LP (15) is only feasible for point F whereas if we replace Eqs. (15b) and (15c) by Eq. (16), the resulting LP will also be feasible for point E.

Remark 2 Alternatively, we can also only restrict p to be a point on the facet-containing hyperplane. In other words, point j merely has to be on the correct side of the facet-containing hyperplane. For this, we need to replace Eqs. (15b) – (15d) by

$$p_k = b^0_{fk} + \sum_{k'=1}^{K-1} \alpha_{k'}\, b_{fk'k} \quad \forall k \tag{16}$$

which would make also point E feasible, as illustrated in Fig. 14b. However, as shown in Fig. 15, this may result in different and often rather odd contours. Therefore, we choose to use the formulation given in (15) in our implementation.

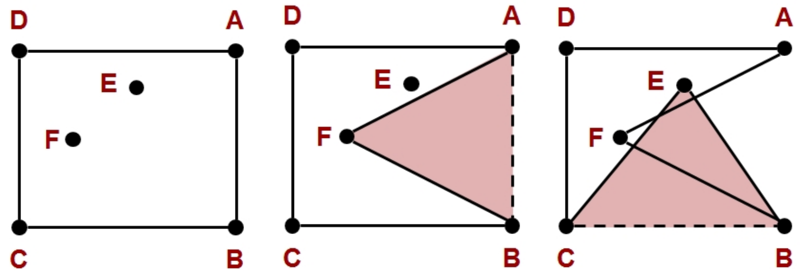


(a) From facet **(b)** From facet-containing hyperplane

Fig. 15: The two different formulations may result in different contours.

Through Eqs. (15f) – (15i), we check every data point j' that is not point j or a vertex of facet f, i.e. $j' \notin H_f^t$, $j' \neq j$, in order to see if the data point lies in the interior of polytope PT_{fj}. Eqs. (15f) and (15g) ensure that $a_{j'}$ is a convex combination of a_j and $a_{j''}$ for $j'' \in H_f^t$ if the corresponding slack variables are zero. In that case, $a_{j'}$ is guaranteed to be in the interior of polytope PT_{fj} because the multipliers are constrained to be greater than or equal to a small value $\bar{\varepsilon}$ in Eq. (15h). At the optimal solution of (15), $\sum_k \left(s^+_{j'k} + s^-_{j'k}\right) = 0$ if point j' lies in the interior of polytope PT_{fj}, in which case we know that point j is not a candidate point for facet f.

However, in order to detect the candidate points, only solving (15) is not sufficient because we also have to consider the situation illustrated in the example shown in Fig. 16. Here, point F is the new vertex originating from facet A-B and one can see that the polytope formed by points A, B and F does not contain any data points in its interior. When we now move on to the next facet and examine facet B-C, we see that one can form a polytope with points B, C and E such that the polytope does not contain any data points in its interior. However, one can clearly see that the formed polytope B-C-E is not fully contained in the current approximation of the feasible region. Therefore, E cannot be a candidate point for facet B-C.



(a) Initial convex hull **(b)** New vertex F from A-B **(c)** E not candidate point for B-C

Fig. 16: F is the new vertex originating from facet A-B. E is not a candidate point for facet B-C because the resulting polytope B-C-E and the already cut-off polytope A-B-F overlap.

Essentially, we have to check if polytope PT_{fj} intersects any of the polytopes that have been "cut off" in previous iterations. For instance, in our example in Fig. 16, we have to check if the polytopes B-C-E and A-B-F overlap each other. We achieve this by solving the following LP:

$$
\begin{aligned}
\min \quad & \sum_{q \in Q} \sum_{k} \left(\hat{s}^{+}_{qk} + \hat{s}^{-}_{qk} \right) && && \text{(17a)} \\
\text{s.t.} \quad & \bar{p}_{qk} = \mu_{jq} a_{jk} + \sum_{j' \in H^t_f} \mu_{j'q} a_{j'k} && \forall q \in Q, k && \text{(17b)} \\
& \mu_{jq} + \sum_{j' \in H^t_f} \mu_{j'q} = 1 && \forall q \in Q && \text{(17c)} \\
& \mu_{j'q} \geq \bar{\varepsilon} && \forall q \in Q, j' = j \text{ or } \in H^t_f && \text{(17d)} \\
& \bar{p}_{qk} = \sum_{j' \in \bar{H}_q} \gamma_{qj'} a_{j'k} + \hat{s}^{+}_{qk} - \hat{s}^{-}_{qk} && \forall q \in Q, k && \text{(17e)} \\
& \sum_{j' \in \bar{H}_q} \gamma_{qj'} = 1 && \forall q \in Q && \text{(17f)} \\
& \gamma_{qj'} \geq \bar{\varepsilon} && \forall q \in Q, j' \in \bar{H}_q && \text{(17g)} \\
& \hat{s}^{+}_{j'k}, \hat{s}^{-}_{j'k} \geq 0 && \forall q \in Q, k && \text{(17h)}
\end{aligned}
$$

where Q is the set of cut-off polytopes and $\bar{H}_q$ is the set of vertices of cut-off polytope q. By solving (17), we try to find a point $\bar{p}_q$ for each $q \in Q$ such that $\bar{p}_q$ is in the interior of polytope PT_{fj} and is also in the interior of polytope q. If we can find such a point for at least one of the cut-off polytopes, point j cannot be a candidate point.

Constraints (17b) – (17d) force $\bar{p}_q$ to be a point in the interior of polytope PT_{fj}. Through Eqs. (17e) – (17h), $\bar{p}_q$ will also be a point in the interior of polytope q if the corresponding slack variables are zero. Since we are minimizing the sum of all slack variables, $\sum_k \left(\hat{s}^{+}_{qk} + \hat{s}^{-}_{qk} \right)$ will be driven to zero if polytope PT_{fj} and polytope q overlap. Consequently, if point j has not been ruled out as a potential candidate point after solving (15) and at the optimal solution of (17), $\sum_k \left(\hat{s}^{+}_{qk} + \hat{s}^{-}_{qk} \right) > 0 \; \forall q \in Q$, we declare point j a candidate point for facet f.

6.1.4 Identifying new vertices and creating new facets

With the set of candidate points C_f and the unit-length normal vector $\bar{n}_f$, we can easily find the candidate point which is furthest away from facet f by solving the following MILP:

$$
\begin{aligned}
\max \quad & d && \text{(18a)} \\
\text{s.t.} \quad & p = \sum_{j \in H^t_f} \lambda_j a_j, \quad \sum_{j \in H^t_f} \lambda_j = 1 && \text{(18b)} \\
& \bar{a} = \sum_{j \in C_f} a_j w_j, \quad \sum_{j \in C_f} w_j = 1 && \text{(18c)}
\end{aligned}
$$

$$\bar{a} = p + d\bar{n}_f, \quad d \geq 0 \tag{18d}$$

$$\lambda_j \geq 0,\ w_j \in \{0,1\} \quad \forall j \in H_f^t \tag{18e}$$

where p is a point on facet f, as given by the convex combination of facet f's vertices in (18b). The binary variable w_j is 1 if candidate point a_j is chosen. At the optimal solution, $\bar{a}$ is the candidate point which is furthest away (larget d) from facet f. Fig. 17 illustrates the process for one facet.

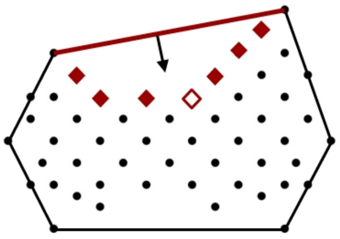


Fig. 17: The data points marked by diamonds are the candidate points for the facet at the top. From all candidate points, the one marked by a hollow diamond is the one furthest away from the facet and is therefore declared as a new vertex.

If d is sufficiently large, i.e. $d \geq \delta$, we add the corresponding point to the set of vertices V. We construct the new facets by connecting the new vertex with the vertices of each $(K-2)$-dimensional facet of facet f. In the 2-dimensional case, a facet of the feasible region is a line and the facets of this facet are always the two end points. Thus, one old facet always leads to two new facets. Note that in higher dimensions, the situation is more complex. The number of new facets created by using one facet and one new vertex is at least K, but could also be significantly larger than K. Finally, with the information on the new facets, we update the sets F, $\bar{F}$, and H_f^{t+1}. Old facets that have been cut off are removed from F while the new facets originated from these old facets are added to F. H_f^{t+1}, the set of vertices for each $f \in F$ in the next iteration $t+1$, is set accordingly. The set of facets to be examined next, $\bar{F}$, only consists of the new facets.

Fig. 18 shows the change in the outer approximation of the feasible region at every iteration. As one can see, it requires three major iterations to obtain the final contour of the feasible region.

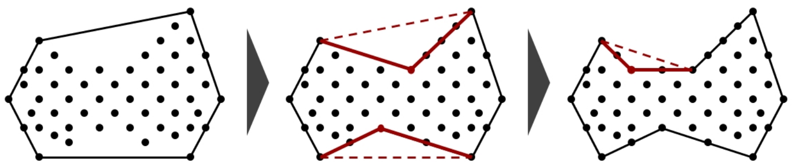


Fig. 18: New facets created at each iteration. The thick red lines show the newly added facets whereas the dashed red lines indicate the facets in the previous iteration from which the new ones originate.

6.2 Constructing convex regions

With the information on the contour of the feasible region, we can now construct the desired convex regions. Fig. 19 shows the flowchart for this part of the Phase 2 algorithm, in which we first set an initial number of regions R, typically to 1. With R fixed, we can then solve the convex region assignment problem, which assigns vertices $j \in V$, non-vertex points $j' \notin V$ and facets $f \in F$ to R convex regions r. If the assignment problem is infeasible, we increase R by 1 and try to solve the problem again.

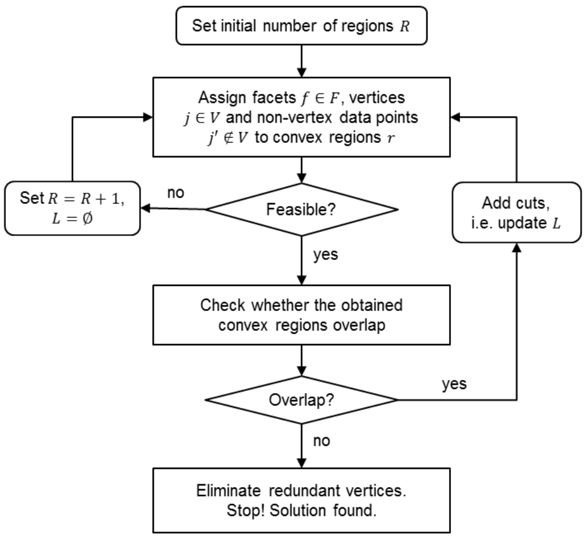

Set initial number of regions R
Assign facets $f \in F$, vertices $j \in V$ and non-vertex data points $j' \notin V$ to convex regions r
Set $R = R + 1$, $L = \emptyset$
no
Feasible?
Add cuts, i.e. update L
yes
Check whether the obtained convex regions overlap
yes
Overlap?
no
Eliminate redundant vertices. Stop! Solution found.

Fig. 19: Flowchart of the algorithm applied to construct the desired convex regions

When the assignment problem is feasible and solved, the algorithm checks whether the obtained convex regions overlap in their interior. If there exist overlapping regions, we add cuts to the assignment problem and solve it again. This process is repeated until we reach a feasible solution which yields convex regions that do not overlap. Finally, after removing redundant vertices, we obtain the desired convex regions described by their vertices.

Remark 3 The motivation to forbidding overlapping convex regions is the reduction of redundant feasible space, in our case space belonging to multiple regions. A point belonging to multiple regions corresponds to multiple equivalent solutions and may slow down the optimization algorithm. However, a CRS model with overlapping convex regions is still a valid surrogate model. Therefore, in some cases, eliminating overlapping convex regions may not be necessary.

6.2.1 Convex region assignment formulation

The convex region assignment problem is the centerpiece of the Phase 2 algorithm. Here, the main idea is to partition the approximation of the feasible region defined by the obtained contour into convex regions, i.e. polytopes. These polytopes are defined by their vertices, which are chosen from the set of vertices V. The union of the convex regions resulting from this partition has to contain all data points.

In order to find these convex regions, we solve an MILP that assigns vertices $j \in V$, non-vertex points $j' \notin V$ and facets $f \in F$ to R convex regions r. The vertices define the polytopes while the assignment of non-vertex points ensures that all data points are contained in the union of the polytopes. The assignment of facets plays a crucial role in the MILP formulation. Namely, it enables us to formulate constraints through which it can be guaranteed that the regions obtained from the assignment problem are actually polytopes. We will elaborate more on this idea later in this section.

Assignment of vertices and facets The following constraints assign vertices and facets to regions.

$$\sum_r x_{rf} = 1 \qquad \forall f \tag{19a}$$

$$\sum_f x_{rf} \geq 1 \qquad \forall r \tag{19b}$$

$$x_{rf} - \sum_{f', f' < f} x_{r-1,f'} \leq 0 \qquad \forall r > 1, f \tag{19c}$$

$$x_{rf} \leq y_{rj} \qquad \forall r, f, j \in H_f \tag{19d}$$

$$\sum_{j \in V} y_{rj} \geq K + 1 \qquad \forall r \tag{19e}$$

$$\sum_r y_{rj} \leq 1 + M w_j \qquad \forall j \in V \tag{19f}$$

$$w_j, x_{rf}, y_{rj} \in \{0, 1\} \qquad \forall r, f, j \in V \tag{19g}$$

where the binary variable x_{rf} is 1 if facet f is assigned to region r, y_{rj} is 1 if vertex j is assigned to region r, and w_j is 1 if vertex j is assigned to more than one region.

Eq. (19a) states that a facet can only be assigned to one region, while constraint (19b) ensures that at least one facet is assigned to a region. Constraint (19c) is a symmetry-breaking constraint that enforces facet-to-region assignment in lexicographic order. Constraint (19d) states that if facet f is assigned to region r, all vertices of facet f, i.e. all $j \in H_f$, are also assigned to region r. To ensure that each region fills a K-dimensional volume, the minimum number of vertices assigned to a region is set to $K + 1$ by constraint (19e). Furthermore, constraint (19f) states that a vertex j can only be assigned to more than one region if $w_j = 1$.

Assignment of non-vertex points The following constraints assign non-vertex points to regions such that the points are inside the regions to which they are assigned.

$$\sum_r z_{rj} = 1 \qquad \forall j \notin V \tag{20a}$$

$$a_j = \sum_{j' \in V} \lambda_{rj'j} a_{j'} + s^+_{rj} - s^-_{rj} \qquad \forall r,\, j \notin V \tag{20b}$$

$$\lambda_{rj'j} \leq y_{rj'} \qquad \forall r,\, j' \in V,\, j \notin V \tag{20c}$$

$$\sum_{j' \in V} \lambda_{rj'j} = 1 \qquad \forall r,\, j \notin V \tag{20d}$$

$$s^+_{rj} \leq M(1 - z_{rj}) \qquad \forall r,\, j \notin V \tag{20e}$$

$$s^-_{rj} \leq M(1 - z_{rj}) \qquad \forall r,\, j \notin V \tag{20f}$$

$$\lambda_{rj'j}, s^+_{rj}, s^-_{rj} \geq 0 \qquad \forall r,\, j' \in V,\, j \notin V \tag{20g}$$

$$z_{rj} \in \{0, 1\} \qquad \forall r,\, j \notin V \tag{20h}$$

where the binary variable z_{rj} is 1 if non-vertex point j is assigned to region r.

Eq. (20a) enforces that a non-vertex point j can only be assigned to one region. Constraints (20b) – (20g) ensure that point j is inside the convex region r if j is assigned to r. The idea is that the corresponding slack variables become zero if z_{rj} is 1 so that a_j is constrained to be a convex combination of the vertices of region r.

Constraining regions to be polytopes The major challenge in the construction of the convex regions is to ensure that the obtained regions are in fact polytopes. In order to formulate such constraints for the convex region assignment problem, we make full use of the information from the contour of the feasible region that we obtained in the previous part of the Phase 2 algorithm. The idea is to force all vertices assigned to region r to be on the same side of the hyperplane containing facet f for every facet f that is assigned to region r. To illustrate this idea, we label the vertices in our illustrative example as shown in Fig. 20.

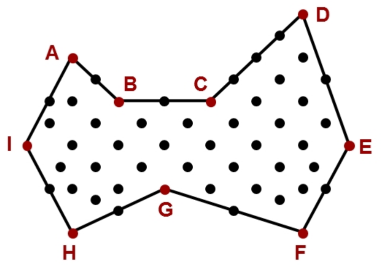
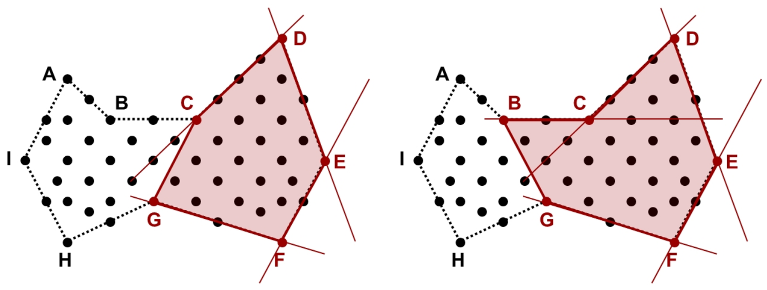


Fig. 20: Contour of the feasible region with labeled vertices

Suppose we assign the facets C-D, D-E, E-F, F-G and the corresponding vertices C, D, E, F, G to the same region. Clearly, they form a convex region, as shown in Fig. 21a. One can see that each facet-containing hyperplane is a supporting hyperplane for the formed convex region, i.e. all points in the region lie on the same side of the hyperplane. If we now want to also include facet B-C and vertex B in the region, the region is not convex anymore, as shown in Fig. 21b. Here, both the hyperplanes corresponding to facet B-C and C-D cut through the region. Therefore, this solution is infeasible for the convex region assignment problem. Since each convex region is defined by its vertices, instead of considering all data points assigned to the region, we only have to constrain all vertices assigned to the region to be on the same side of the corresponding facet-containing hyperplanes.



(a) Formed region is convex. **(b)** Formed region is nonconvex.

Fig. 21: A region is convex if and only if the hyperplanes containing the assigned facets are supporting hyperplanes for the region.

The resulting constraints are the following:

$$p_{rfj} = b_f^0 + \sum_{k'=1}^{K-1} \alpha_{rfjk'} b_{fk'} + \tilde{s}^+_{rfj} - \tilde{s}^-_{rfj} \qquad \forall r, f, j \in V \tag{21a}$$

$$a_j = p_{rfj} + d_{rfj} \bar{n}_f + \hat{s}^+_{rfj} - \hat{s}^-_{rfj} \qquad \forall r, f, j \in V \tag{21b}$$

$$\tilde{s}^+_{rfj} \leq M\left(1 - x_{rf}\right) \qquad \forall r, f, j \in V \tag{21c}$$

$$\tilde{s}^-_{rfj} \leq M\left(1 - x_{rf}\right) \qquad \forall r, f, j \in V \tag{21d}$$

$$\hat{s}^+_{rfj} \leq M(1 - y_{rj}) \qquad \forall r, f, j \in V \tag{21e}$$

$$\hat{s}^-_{rfj} \leq M(1 - y_{rj}) \qquad \forall r, f, j \in V \tag{21f}$$

$$d_{rfj}, \tilde{s}^+_{rfj}, \tilde{s}^-_{rfj}, \hat{s}^+_{rfj}, \hat{s}^-_{rfj} \geq 0 \qquad \forall r, f, j \in V \tag{21g}$$

Eq. (21a) indicates that p_{rfj} is a point on the hyperplane containing facet f if the slack variables $\tilde{s}^+_{rfj}$ and $\tilde{s}^-_{rfj}$ are zero, which has to hold true if facet f is assigned to region r, i.e. $x_{rf} = 1$, as enforced by constraints (21c) and (21d). In Eq. (21b), d_{rfj} denotes a nonnegative distance and $\bar{n}_f$ is the vector normal to facet f and pointing towards the interior of the feasible region. Constraints (21b), (21e) and (21f) imply that vertex j has to be on the same side of the hyperplane containing facet f as the formed region r if vertex j is assigned to region r.

Objective function The MILP minimizes

$$\sum_r \sum_{j \in V} y_{rj} + \sum_{j \in V} w_j \tag{22}$$

which is the total number of vertex-to-region assignments and the total number of vertices that are assigned to more than one region. This is a heuristic used to discourage the construction of overlapping convex regions.

The full convex region assignment problem is then:

$$\begin{aligned} \min \quad & (22) \\ \text{s.t.} \quad & \text{Eqs. (19), (20), (21)} \end{aligned} \tag{23}$$

6.2.2 Identifying and eliminating overlapping convex regions

The optimal solution of the convex region assignment problem may yield convex regions that overlap each other in their interior, as illustrated in the first sketch in Fig. 22. The following LP, which we use to identify overlapping regions, is almost identical to (9) used in Phase 1. Except, here we restrict the multipliers $\lambda_{rr'j}$ and $\mu_{rr'j}$ to be greater than a small value $\bar{\varepsilon}$ because we want to detect regions which overlap in their interior. Regions that only overlap at the boundary are not considered overlapping regions.

$$\min \quad \sum_{r} \sum_{r', r<r'} \sum_{k} \left(s^{+}_{rr'k} + s^{-}_{rr'k}\right) \tag{24a}$$

$$\text{s.t.} \quad p_{rr'k} = \sum_{j \in V_r} \lambda_{rr'j} a_{jk} \qquad \forall r, r', r<r', k \tag{24b}$$

$$p_{rr'k} = \sum_{j \in V_{r'}} \mu_{rr'j} a_{jk} + s^{+}_{rr'k} - s^{-}_{rr'k} \qquad \forall r, r', r<r', k \tag{24c}$$

$$\sum_{j \in V_r} \lambda_{rr'j} = \sum_{j \in V_{r'}} \mu_{rr'j} = 1 \qquad \forall r, r', r<r' \tag{24d}$$

$$\lambda_{rr'j} \geq \bar{\varepsilon} \qquad \forall r, r', r<r', j \in V_r \tag{24e}$$

$$\mu_{rr'j} \geq \bar{\varepsilon} \qquad \forall r, r', r<r', j \in V_{r'} \tag{24f}$$

$$s^{+}_{rr'k}, s^{-}_{rr'k} \geq 0 \qquad \forall r, r', r<r', k \tag{24g}$$

If overlapping regions exist, we can use the solution of (24) to generate cuts equivalent to the ones in (10) described in Section 5.4 and add them to the convex region assignment problem. By repeatedly resolving (23) with added cuts if overlapping regions are detected, we converge to a solution with no overlap, such as the one illustrated in the second sketch of Fig. 22.

6.2.3 Removing redundant vertices

The algorithm may declare data points vertices although they may later be assigned to convex regions in which they are not vertices of the formed polytopes. In the example shown in Fig. 23, point B has been declared a vertex in Phase 1. However, in Phase 2, we find the two polytopes shown in Fig. 23b which provide an accurate approximation of the feasible region. Point B is on a facet of one of the two polytopes but is not a vertex. Hence, point B is redundant because it is not needed for describing the polytope.

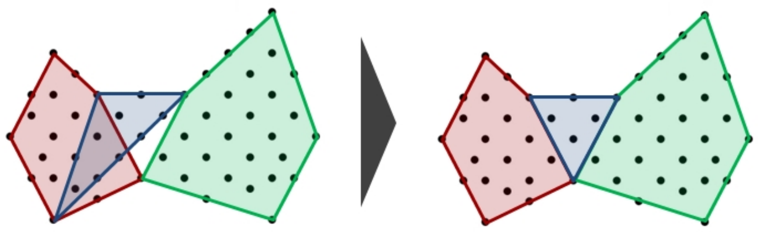
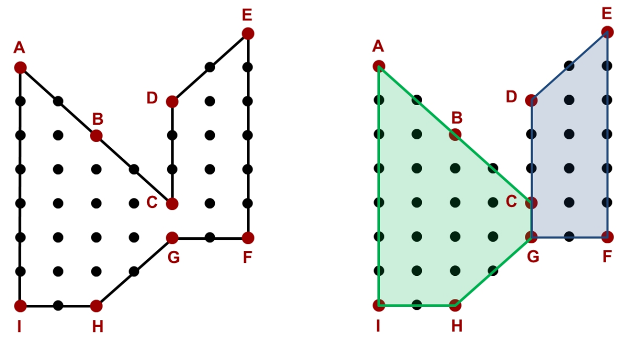


Fig. 22: The convex region assignment problem may yield overlapping convex regions. By adding the proposed cuts, these solutions can be avoided so that we obtain the desired convex regions when the algorithm terminates.



(a) This contour of the feasible region is obtained in Phase 1.

(b) Phase 2 terminates after finding these two convex regions.

Fig. 23: In this example, point B is a redundant vertex because it is declared a vertex in Phase 1, but is not a vertex of any of the convex regions constructed in Phase 2.

We can eliminate these redundant vertices by finding the vertices of each polytope and removing the ones that are not true vertices. To find the vertices, we can again apply the Quickhull algorithm or use the MILP given in (7). The whole algorithm terminates after this step since now we have obtained all convex regions with the corresponding vertices required to build our CRS model.

7 Summary of algorithm

To summarize, we show the complete CRS algorithm at one glance. Note that for the description of the Phase 2 algorithm, we reintroduce the subset index i, which we omitted for the sake of readability in Section 6.

Phase 1 (Subset assignment with cost correlation constraints)

Step 1.1 (Initialization) Set iteration counter $t = 1$, initial number of subsets $m = 1$, and specify error tolerance ε.

Step 1.2 (Subset assignment) Construct set of subsets $I^t = \{1, \dots, m\}$, and solve (5) or (6) to assign data points to subsets. If feasible, construct the sets of assigned data points J_i, and go to Step 1.3. Otherwise, set $t = t+1$, $m = m+1$, reset the set of overlap elimination cuts $S = \emptyset$, and repeat Step 1.2.

Step 1.3 (Constructing convex hulls) Apply the Quickhull algorithm to or solve (7) for each subset $i \in I^t$ to obtain the sets of vertices V_i for the convex hulls around the data points assigned to each subset.

Step 1.4 (Identifying overlapping convex hulls and generating cuts) Check for overlapping convex hulls by solving (9). If no overlap, construct sets of facets F_i and sets of facet vertices H_{if}, and go to Step 2.1. Otherwise, generate cuts as given by (10), add the cuts to (5) or (6), and return to Step 1.2.

Phase 2 (Convex region assignment)

Step 2.1 (Starting loop over set of subsets) Set subset index $i = 1$, specify minimum distances δ.

Step 2.2 (Initializing contour construction) Set iteration counter $t = 1$, the set of facets to be examined $\bar{F} = F_i$, and the set of facet vertices $H^t_{if} = H_{if}$.

Step 2.3 (Construction of contour of feasible region)

Step 2.3.1 (Starting loop over set of facets) Set f to be the first element in $\bar{F}$.

Step 2.3.2 (Obtaining normal vector) Obtain a vector n_f normal to facet f as described in Section 6.1.1. Determine the direction in which n_f points by solving (12) if $t = 1$ or (13) if $t > 1$. Set $\bar{n}_f = n_f$ if n_f points towards the interior of the feasible region, set $\bar{n}_f = -n_f$ if otherwise.

Step 2.3.3 (Checking for point in facet's interior) Solve (14). If no data point found in the interior of facet f, set set of candidate points $C = \emptyset$, and go to Step 2.3.4a. Otherwise, go to Step 2.3.6.

Step 2.3.3 (Finding candidate points for new vertex)

Step 2.3.4a Set j to be the first element of $P = \{j : j \in J_i \wedge j \notin V_i\}$.

Step 2.3.4b Solve (15). If feasible and no point detected in the interior of polytope PT_{fj}, go to Step 2.3.4c. Otherwise, go to Step 2.3.4d.

Step 2.3.4c Solve (17). If no overlap between polytope PT_{fj} and any cut-off polytopes $q \in Q$, add j to C.

Step 2.3.4d If j is the last element in P, go to Step 2.3.5. Otherwise, set j to the next element in P and return to Step 2.3.4b.

Step 2.3.5 (Determining new vertex and facets) If $C \neq \emptyset$, solve (18), obtain distance d and the corresponding point j. If $d \geq \delta$, add point j to the set of vertices V_i, create new facets, remove facet f from and add new facets to F_i, add cut-off polytope to Q and update set of cut-off polytope vertices $\bar{H}_{iq}$.

Step 2.3.6 (Closing loop over set of facets) If f is the last element in $\bar{F}$, go to Step 2.3.7. Otherwise, set f to the next element in $\bar{F}$ and return to Step 2.3.2.

Step 2.3.7 (Updating contour information) Construct H^{t+1}_{if} according to F_i. If new facets have been created, empty $\bar{F}$ and add the new facets to it, set $t = t+1$, and return to Step 2.3.1. Otherwise, go to Step 2.4.

Step 2.4 (Initializing convex region assignment) Set initial number of regions $R = 1$.

Step 2.5 (Convex region assignment) Solve convex region assignment problem (23). If feasible, go to Step 2.6. Otherwise, set $R = R + 1$, reset the set of overlap elimination cuts $L = \emptyset$, and repeat Step 2.5.

Step 2.6 (Identifying overlapping regions and generating cuts) Check for overlapping convex regions by solving (24). If no overlap, go to Step 2.7. Otherwise, generate cuts, add the cute to (23), and return to Step 2.5.

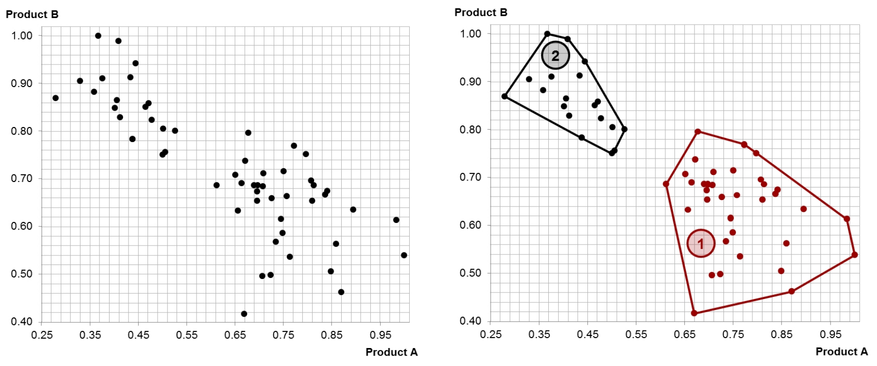
Step 2.7 (Removing redundant vertices) Remove redundant vertices as described in Section 6.2.3. If $i = m$, stop and report the solution in form of the parameter values VT_{irjk} corresponding to subset i, convex region r, vertex j, and dimension k. If $i < m$, set $i = i + 1$ and return to Step 2.2.

The algorithm is implemented in MATLAB[1] and GAMS[2] in order to make use of the Quickhull algorithm in MATLAB and the optimization solvers in GAMS.

8 Case study

We now apply the proposed algorithm to a real-world industrial case study provided by Praxair. Here, the objective is to construct a CRS model of a production process that can be integrated into a scheduling optimization problem, which is formulated as an MILP. In this case, we can limit the variable space of the CRS model to two dimensions since we are solely interested in the production rates of Product A and Product B. The objective function is the power consumption.

We are given 55 data points directly drawn from the actual process. Each data point carries production rate values for Product A and Product B as well as the corresponding power consumption value. The data are plotted in the 2-dimensional product space in Fig. 24a and listed in Table 1. Note that for this case study, we use normalized values.



(a) Given data points **(b)** Subset assignment result

Fig. 24: In Phase 1, the given data points are partitioned into two disjoint subsets, denoted subsets 1 and 2.

[1] MATLAB version R2012a (7.14.0.739), The Mathworks Inc.

[2] GAMS version 24.0.2, GAMS Development Corporation

Table 1: Production rate and power consumption data from the production process

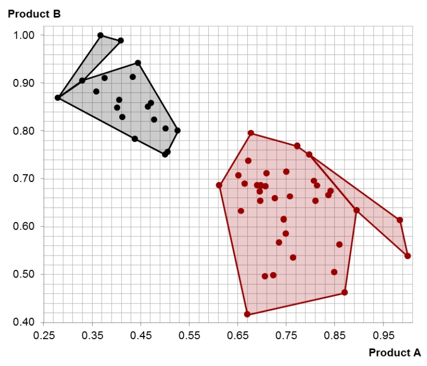
A	0.438	0.772	0.650	0.797	0.841	0.707	0.611	0.695	0.500	0.750
B	0.783	0.769	0.708	0.751	0.675	0.685	0.686	0.673	0.751	0.715
Power	0.961	0.934	0.959	0.931	0.952	0.951	0.958	0.957	0.961	0.956
A	0.813	0.836	0.848	0.807	0.706	0.502	0.412	0.859	1.000	0.444
B	0.686	0.666	0.505	0.696	0.496	0.804	0.829	0.563	0.539	0.942
Power	0.958	0.957	0.971	0.923	0.956	0.950	0.955	0.953	0.979	0.992
A	0.409	0.367	0.984	0.656	0.677	0.434	0.471	0.895	0.669	0.763
B	0.989	1.000	0.614	0.633	0.796	0.913	0.858	0.635	0.417	0.536
Power	0.993	1.000	0.977	0.958	0.934	0.957	0.958	0.940	0.921	0.938
A	0.749	0.664	0.723	0.735	0.505	0.689	0.726	0.757	0.405	0.477
B	0.586	0.690	0.498	0.567	0.756	0.686	0.660	0.663	0.865	0.823
Power	0.936	0.941	0.941	0.917	0.945	0.955	0.952	0.956	0.948	0.943
A	0.744	0.697	0.465	0.527	0.671	0.708	0.870	0.809	0.329	0.401
B	0.616	0.687	0.850	0.801	0.738	0.711	0.463	0.654	0.905	0.849
Power	0.921	0.948	0.954	0.952	0.931	0.955	0.965	0.928	0.942	0.985
A	0.376	0.358	0.696	0.279	0.438					
B	0.910	0.882	0.654	0.869	0.783					
Power	0.983	0.940	0.941	0.938	0.961					

For the subset assignment in Phase 1, we set the maximum error tolerance ε to 3 %. This results in the partitioning of the given data points into two disjoint subsets. Fig. 24b shows the assignment of data points to the two subsets and the resulting convex hulls. The corresponding cost constants and coefficients for the linear power consumption correlations are shown in Table 2.

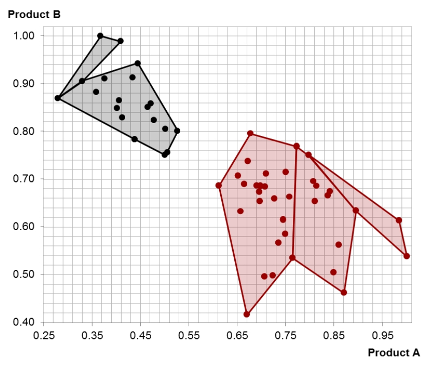
Table 2: Constants and coefficients for the power consumption correlations

Subset	b	c_A	c_B
1	0.900	0.062	0.000
2	0.703	0.127	0.236

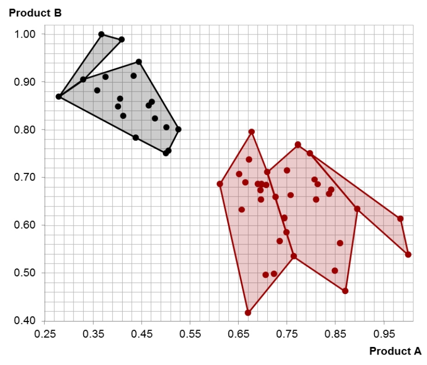
The result from Phase 2 strongly depends on the specified δ, the minimum distance between a new vertex and the facet that it originates from. For comparison, we run the Phase 2 algorithm for four different δs, for which the resulting convex regions are shown in Fig. 25. One can see that the smaller δ is, the more points tend to be declared vertices, which leads to the construction of more convex regions. While for $\delta = 0.10$, 4 convex regions are constructed, we obtain 11 convex regions for $\delta = 0.04$. For $\delta = 0.08$ and $\delta = 0.06$, we have the same number of convex regions. However, the convex regions for $\delta = 0.06$ describe a tighter envelope around the data points.



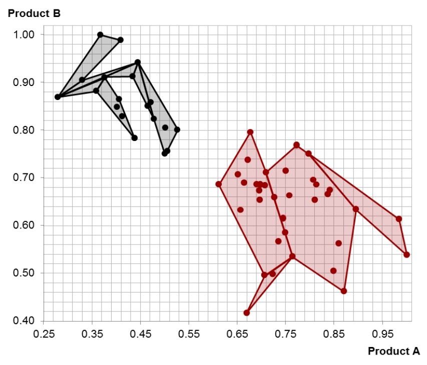
(a) $\delta = 0.10$



(b) $\delta = 0.08$



(c) $\delta = 0.06$



(d) $\delta = 0.04$

Fig. 25: Depending on δ, the Phase 2 algorithm constructs different convex regions.

This example also nicely illustrates a key principle in data-driven approaches: A good method does not necessarily lead to good results if it is not given a sufficiently large set of useful data. Clearly, among the cases shown in Fig. 25, $\delta = 0.04$ results in the tightest envelope around all data points. However, this may not provide the most accurate approximation of the feasible region. Especially when observing the shape of the union of the convex regions for Subset 2, there is reason to suspect that this does not resemble the actual feasible region of the process. However, if we had more data points filling up a larger portion of the feasible space, we might obtain a different result or at least, we would be more confident about the obtained result. In this case study, due to the limited number of data points, we would rather apply the CRS model resulting from $\delta > 0.04$, e.g $\delta = 0.06$.

The algorithm was performed on an Intel i7 machine at 3.4 GHz with 8 GB RAM. Phase 1 was completed in 7 seconds. For Phase 2, we report the computational results for the two extreme cases. For $\delta = 0.10$, 347 LPs, 20 NLPs and 38 MILPs were solved, and the total CPU time was 70 seconds. For $\delta = 0.04$, 567 LPs, 36 NLPs and 75 MILPs were solved, and the total CPU time was 135 seconds. The computational bottleneck was the largest convex region assignment problem, which in the case of

$\delta = 0.04$ was an MILP with 17,556 constraints, 17,095 continuous variables and 245 binary variables.

9 Conclusions

This work addresses the challenge of generating accurate and computationally efficient surrogate models for mixed-integer programming frameworks. In this paper, we have introduced the idea of Convex Region Surrogate models, which can be formulated as MILPs yet still provide good approximations of nonlinearities and nonconvexities. In a CRS model, the feasible region is approximated by the union of convex regions in the form of polytopes. Furthermore, for each region, the objective function is approximated by a linear function.

We have presented a two-phase algorithm for the construction of CRS models based on given data. In Phase 1, the algorithm partitions the data points into disjoint subsets such that a linear cost correlation can be obtained within en error tolerance for all points in each subset, and that the convex hulls around the points of each subset do not overlap. In Phae 2, convex regions are constructed for each subset such that the union of the convex regions describes an accurate approximation of the feasible region. The algorithm is presented in detail in the main part of this paper.

We have further applied the proposed algorithm to an industrial test case drawn from a Praxair plant. In the case study, data from a real production process have been used to generate a CRS model with a feasible region in the product space and power consumption as the objective function. The case study has demonstrated the effectiveness of the algorithm as well as provided insights into the impact of the given data on the result from the algorithm

Acknowledgements The authors gratefully acknowledge the financial support from the National Science Foundation under grant # 1159443. The authors further want to thank Narcisse Pokem Kamgui for his input on the case study.

Appendix: Data for Illustrative Example

For the illustrative example used throughout this paper, we have constructed the set of 100 data points shown in Table 3. Each data point consists of a 2-dimensional parameter vector a and a corresponding cost value g. The cost values are calculated from the parameter values using two different linear correlations, for which the corresponding constants and coefficients are listed in Table 4. The cost values for the first 52 points are generated from the first linear correlation while the cost values for the remaining points are calculated by applying the second linear correlation.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software 22(4):469–483

Table 3: Complete set of data for the illustrative example

a_1	4	14	6	11	5	13	3	4	6	8
a_2	7	7	8	8	8.5	8.5	9	9	9	9
g	39	59	46	56	45.5	61.5	43	45	49	53
a_1	10	12	14	15	3.5	5	7	9	11	13
a_2	9	9	9	9	10	10	10	10	10	10
g	57	61	65	67	47	50	54	58	62	66
a_1	14.5	2	4	6	8	10	12	14	16	3
a_2	10	11	11	11	11	11	11	11	11	12
g	69	47	51	55	59	63	67	71	75	52
a_1	5	7	9	11	13	15	3	4	6	8
a_2	12	12	12	12	12	12	13	13	13	13
g	56	60	64	68	72	76	55	57	61	65
a_1	10	12	14	5	11	13	15	4	12	14
a_2	13	13	13	14	14	14	14	15	15	15
g	69	73	77	62	74	78	82	63	79	83
a_1	13	14	4	6	8	10	12	5	7	9
a_2	16	17	2	2	2	2	2	3	3	3
g	84	89	320	380	440	500	560	400	460	520
a_1	11	3	4.5	6	8	10	11.5	13	3.5	5
a_2	3	4	4	4	4	4	4	4	5	5
g	580	390	435	480	540	600	645	690	455	500
a_1	7	9	11	12.5	2	4	6	8	10	12
a_2	5	5	5	5	6	6	6	6	6	6
g	560	620	680	725	460	520	580	640	700	760
a_1	14	6	8	7	10	9	12	7	10	14
a_2	6	16	16.5	17	17	17.5	17.5	18	18	18
g	820	1080	1165	1160	1250	1245	1335	1210	1300	1420
a_1	8	12	9	11	13	14	8	10	12	14
a_2	18.5	18.5	19	19	19	19	20	20	20	20
g	1265	1385	1320	1380	1440	1470	1340	1400	1460	1520

Table 4: Cost constants and coefficients used in the illustrative example

	b	c_1	c_2
First 52 points	10	2	3
Remaining 48 points	100	30	50

Biegler LT, Lang Yd, Lin W (2014) Multi-scale optimization for process systems engineering. Computers & Chemical Engineering 60:17–30

Chung PS, Jhon MS, Biegler LT (2011) Chapter 2 - The Holistic Strategy in Multi-Scale Modeling. Advances in Chemical Engineering 40:59–118

Crowder H, Johnson EL, Padberg M (1983) Solving Large-Scale Zero-One Linear Programming Problems. Operations Research 31(5):803–834

GAMS Development Corporation (2013) GAMS version 24.0.2

Goyal V, Ierapetritou MG (2002) Determination of operability limits using simplicial approximation. AIChE Journal 48(12):2902–2909

Goyal V, Ierapetritou MG (2003) Framework for evaluating the feasibility/operability of nonconvex processes. AIChE Journal 49(5):1233–1240

Grossmann IE, Trespalacios F (2013) Systematic Modeling of Discrete-Continuous Optimization Models through Generalized Disjunctive Programming. AIChE Journal 59(9):3276–3295

Ierapetritou MG (2001) New approach for quantifying process feasibility: Convex and 1-D quasi-convex regions. AIChE Journal 47(6):1407–1417

Jain AK (2010) Data clustering: 50 years beyond K-means. Pattern Recognition Letters 31(8):651–666

Jain AK, Duin RPW, Mao J (2000) Statistical Pattern Recognition : A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(1):4–37

Karwan MH, Keblis MF (2007) Operations planning with real time pricing of a primary input. Computers & Operations Research 34(3):848–867

Mitra S, Grossmann IE, Pinto JM, Arora N (2012) Optimal production planning under time-sensitive electricity prices for continuous power-intensive processes. Computers & Chemical Engineering 38:171–184

Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Kevin Tucker P (2005) Surrogate-based analysis and optimization. Progress in Aerospace Sciences 41(1):1–28

Simpson TW, Peplinski JD, Koch PN, Allen JK (2001) Metamodels for Computer-based Engineering Design : Survey and recommendations. Engineering with Computers 17:129–150

Sung C, Maravelias CT (2007) An Attainable Region Approach for Production Planning of Multiproduct Processes. AIChE Journal 53(5):1298–1315

Sung C, Maravelias CT (2009) A Projection-Based Method for Production Planning of Multiproduct Facilities. AIChE Journal 55(10):2614–2630

Swaney RE, Grossmann IE (1985) An Index for Operational Flexibility in Chemical Process Design - Part I: Formulation and Theory. AIChE Journal 31(4):621–630

The Mathworks Inc (2012) MATLAB R2012a (7.14.0.739)

Üney F, Türkay M (2006) A mixed-integer programming approach to multi-class data classification problem. European Journal of Operational Research 173(3):910–920

Wang GG, Shan S (2007) Review of Metamodeling Techniques in Support of Engineering Design Optimization. Journal of Mechanical Design 129(4):370