J Glob Optim (2007) 39:577–593
DOI 10.1007/s10898-007-9157-x

Infeasibility spheres for finding robust solutions of blending problems with quadratic constraints

Leocadio G. Casado · Eligius M. T. Hendrix · Inmaculada García

Received: 6 March 2007 / Accepted: 17 March 2007 / Published online: 6 June 2007

Abstract The blending problem is studied as a problem of finding cheap robust feasible solutions on the unit simplex fulfilling linear and quadratic inequalities. Properties of a regular grid over the unit simplex are discussed. Several tests based on spherical regions are described and evaluated to check the feasibility of subsets and robustness of products. These tests have been implemented into a Branch-and-Bound algorithm that reduces the set of points evaluated on the regular grid. The whole is illustrated numerically.

Keywords Blending · Branch-and-Bound · Quadratic programming · Robust solutions

1 Introduction

Consider the following formulation of a mixture design problem which actually consists of identifying mixture products, each represented by a vector $x \in \mathbb{R}^n$, which meet certain requirements. The set of possible mixtures is mathematically defined by the unit simplex

$$S = \left\{ x \in \mathbb{R}^n \mid \sum_{j=1}^{n} x_j = 1.0;\ 0 \le x_j \le 1 \right\}, \tag{1}$$

L. G. Casado · I. García
Departmento de Arquitectura de Computadores y Electrónica, Universidad de Almeria, Almeria, Spain
e-mail: leo@ace.ual.es

I. García
e-mail: igarcia@ual.es

E. M. T. Hendrix (✉)
Operationele Research en Logistiek Groep, Wageningen Universiteit, Wageningen, The Netherlands
e-mail: eligius.hendrix@wur.nl

where the variables x_j represent the fraction of the components in a product x. In mixture design (blending) problems, the objective is to minimise the cost of the material,

$$f(x) = c^T x, \tag{2}$$

where vector c gives the cost of the raw materials. In practical situations, such problems are solved on a daily base in fodder and petrochemical industry where often requirements are modelled by linear inequalities, see e.g., [15]. The current article is a result from a larger project on product design at Unilever Research. Products that are produced in large quantities require extensive testing and careful designing where many aspects such as robustness, cost, choice and availability of raw materials, etc. are important. Here, we focus on methods to generate so-called ϵ-robust solutions, i.e., small deviations from the design will still lead to acceptable products.

In the model under study, linear inequality constraints and bounds define the design space $X \subset S$. The requirements are defined as quadratic inequalities

$$g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 1, \ldots, m, \tag{3}$$

in which A_i is a symmetric n by n matrix, b_i is an n-vector and d_i is a scalar. In this way we formulate the problem to be solved as finding elements of the set of "satisfactory" (feasible) products

$$D = \{x \in S \mid g_i(x) \leq 0; \quad i = 1, \ldots, m\}. \tag{4}$$

Finding a point $x \in X \cap D$ defines the quadratic mixture design problem (QMDP), as studied in [6].

In the practical setting the problem originates from, besides optimising the linear objective function, the search for feasible solutions can be approached by running standard solvers on a penalty reformulation:

$$\min_{x \in X} \{\max_i g_i(x)\} \tag{5}$$

or

$$\min_{x \in X} \left\{ \sum_i \max[g_i(x), 0] \right\}. \tag{6}$$

The challenging aspect is that solving (5) or (6) is a global optimisation problem. This means that applying a standard solver, using several starting points may result into local optima that are all bigger than zero, i.e., no feasible solution of (3) is attained. In that case, one cannot assure that no feasible solution exists.

From practical considerations, this problem was extended towards robust solutions. One can define robustness $R(x)$ of a design $x \in D$ with respect to D as

$$R(x) = \max\{R \in \mathbb{R}^+ \mid (x + h) \in D, \ \forall h \in \mathbb{R}^n, \ \|h\| \leq R\}. \tag{7}$$

Notice that for mixture problems $x + h$ is projected on the unit simplex. In [7] an analytical expression of the robustness for linear mixture design problems is given and it is shown that for quadratic inequalities this is not straightforward. One should find the nearest infeasible point to x and no analytic expression exists to determine this point, such that algorithms as described in [4] are needed. Moreover, [7] shows that determining the point with maximum robustness $R(x)$ becomes a Global optimisation problem. We are merely interested in methods for finding an ϵ-robust solution with minimum cost, i.e.

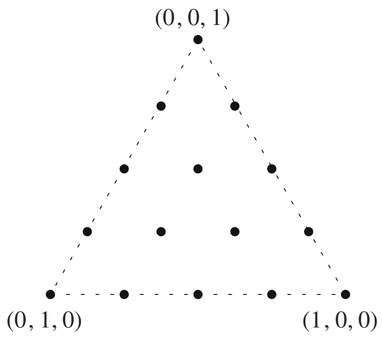

Fig. 1 Two-dimensional projection of regular grid over the three-dimensional unit simplex, $M = 5$

$$\begin{array}{lll} \min & f(x) & \text{(Cost} \quad (2)) \\ \text{s.t.} & x \in X \cap D & \text{(Feasibility} \quad (4)) \\ & R(x) \geq \epsilon & \text{(Robustness} \quad (7)). \end{array} \tag{8}$$

Moreover, we would like to develop an algorithm that is able to identify problems that do not contain ϵ-robust solutions. The developed algorithm is based on the Branch-and-Bound concept applying simplicial partition sets. Several options are described and tested on how to check feasibility of a subset and on how to determine lower bounds on the robustness. Theoretical results are presented with respect to areas of the search space that do not contain a feasible point.

In Sect. 2, the algorithm is sketched and the used bisection process is analysed. Results are given on bounds on the number of points that are evaluated. In Sect. 3, ways to test the infeasibility of partition sets are described and theoretical results are presented. Section 4 describes ways to check the ϵ-robustness. Numerical illustrations with cases derived from industry are shown in Sect. 5. Finally conclusions are drawn in Sect. 6.

2 Algorithms for finding a solution

One approach, as sketched before, is to run nonlinear optimisation routines on penalty formulation (5) or (6) meanwhile minimising the linear objective function (2). However, if the final penalty answer is greater than zero ($g_i(x) > 0$), the statement "no solution exists" cannot be given with absolute certainty. Given a certain accuracy, one can guarantee to detect cases that do not contain solutions. This is done by evaluating all the points on a regular grid over the unit simplex as sketched in Fig. 1. As will be outlined in Sect. 3, one can prove that no feasible solution exists, if the obtained function values g_i are "high enough". Moreover, we can also conclude that there is no ϵ-robust solution when every evaluated feasible point has an infeasible neighbour at a distance less than ϵ on the regular grid. However, checking robustness requires additional procedures as those described in Sect. 4.

Consider a regular grid with M equal distant values for each axis, resulting in a mesh size of $\alpha = 1/(M - 1)$. A strategy to evaluate all grid points is not appealing, as it is not efficient. When performed on a unit box, the number of function evaluations grows exponentially with the dimension: M^n. This is not that bad on the unit simplex, as we are dealing with the mixture equality (see (1) and Fig. 1). It can be verified that the total number of points on the grid is given by

$$\sum_{k=1}^{n-1} \binom{M}{k} \binom{n-2}{k-1}, \tag{9}$$

as shown in Appendix 1. This means that the number of points increases rapidly. For the example in Fig. 1, $n = 3$ and $M = 5$, we have 15 grid points and an "accuracy" of $\alpha = 0.25$.

An appealing alternative for solving (8) is to use a Branch-and-Bound strategy. The concept of Branch-and-Bound is not to generate all the points, but to partition the area and avoid visiting those regions (partition sets) which are known not to contain an optimal ϵ-robust solution. B&B methods can be characterized by four rules: *Branching, Selection, Bounding*, and *Elimination* [10,12]. For problems like (8), where the number of points can be infinite, a termination criterion has to be incorporated; i.e, one has to establish a sampling precision.

In the Branch-and-Bound method, the initial problem is subsequently partitioned in more and more refined subproblems (branching) over which bounds of an objective function value, and in our case, bounds on the constraint functions can be determined (bounding). The search is reduced by eliminating subproblems. One of the elimination rules used here is based on defining a global upper bound f^U as the objective function value of the best ϵ-robust solution found so far. Subsets with a lower bound f_k^L of the objective function larger than the upper bound can be discarded, because they cannot contain an optimal solution.

A possible algorithm based on bisection is outlined in Algorithm 1. The method starts with a set $C_1 = S$ as the first element of a list Λ of subsets (partition sets) and stops when the list Λ is empty. A generated subset is not stored on Λ, if it can be proven that it is infeasible and/or cannot contain an ϵ-robust solution. In Sect. 3, theoretical results on proving infeasibility are discussed. Section 4 concerns the determination of robustness of individual points.

The size of a simplex ($Size(C)$) is given by the Euclidian length of its largest edge. To force theoretical convergence, the termination rule establishes that the search does not continue on partition sets smaller in size than $\alpha \leq \epsilon$. If a simplex reaches the termination criterion and all vertices are feasible, but not proven ϵ-robust, one cannot certify anymore that the problem has no ϵ-robust solution. In Algorithm 1, this is traced by a flag "NoSolutionExists" that has the value True when the problem has no ϵ-robust solution, i.e., all the simplices were rejected by one of the elimination rules before the termination criterion was applied to a simplex with feasible vertices.

The branching concerns the further refinement of the partition. This means that one of the subsets is selected to be split into new subsets. The use of simplicial sets in Branch-and-

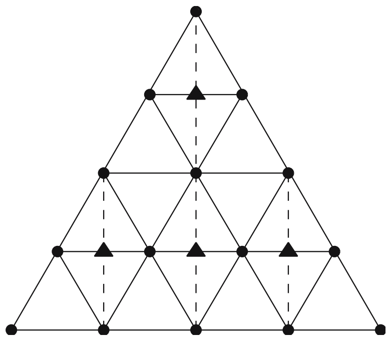
Fig. 2 Bisection process. Two-dimensional projection

Algorithm 1 : Branch-and-Bound algorithm

```
Inputs:   $X$: linearly bounded design space
          $c_j$: cost of component $j$
          $g_i$: constraint functions
          $\epsilon$: accuracy on robustness
          $\alpha$: accuracy of search. ($\alpha \leq \epsilon$)
Output:   $\epsilon$-robust solution $x^U$, "No solution found" or "No solution exists"
Funct B&B Algorithm
 1. $\Lambda := \emptyset$                                        Work list
 2. $f^U := \max_j c_j$                                           Upper bound of the solution
 3. $x^U := 0$                                                    $\epsilon$-robust solution product
 4. $C_1 = S$                                                     Unit simplex
 5. $NoSolutionExists$=True
 6. for the vertices $v \in C_1$ EvaluateVertex($v$)
 7. EvaluateSimplex($C_1$)
 8. while $\Lambda \neq \emptyset$
 9.   Take one subset $C$ from list $\Lambda$ according to a selection rule.
        Subdivide $C$ into two new subsets $Cnew_1$ and $Cnew_2$ by splitting
        over the longest edge, generating new point $v_{new}$.
10.   EvaluateVertex($v_{new}$),
11.   EvaluateSimplex($Cnew_1$), EvaluateSimplex($Cnew_2$)
12. if $x^U = 0$
13.   if $NoSolutionExists$
14.     return "No solution exists"
15.   else
16.     return "No solution found"
17. else return $x^U$
```

Algorithm 2 : Evaluate subset, decide to put on list based on lower bounds

```
Funct EvaluateSimplex ($C_k$); global $\Lambda$, $f^U$, $\alpha \leq \epsilon$, $NoSolutionExists$
1. $f_k^L := \min_{v \in C_k} \{c^T v\}$
2. if $f_k^L \leq f^U$                                            Pruning
3.   Infeasibility check on $C_k$                                 See Sect. 3
4.   if $C_k$ not proven to be infeasible
5.     if $size(C_k) > \alpha$                                    Stopping criteria
6.       store $C_k$ in $\Lambda$
7.     elsif all vertices of $C_k$ are feasible
8.       $NoSolutionExists$=False
```

Bound and several ways of splitting them has been studied extensively in [3,9]. Algorithm 1 bisects the longest edge of the selected simplex. An advantage of this kind of bisection is that the sets never get a needle shape. Starting with the unit simplex, for all the generated simplices the length of the longest edge is at most twice the size of the shortest edge. A selection rule determines the subset to be split next. The cheapest simplex (with lowest average cost) is selected, trying to improve the value of f^U. Figure 2 sketches the idea of the bisection algorithm. It can be observed that points on a regular grid are generated, but that the bisection also generates edges (dotted lines) in at least one additional direction other than the facets of the unit simplex. The values of all generated points are a multiple of $(1/2)^K$, where K is an integer representing the depth of the search tree.

The number of simplices that Algorithm 1 generates (and stores) in the worst case, depends on many aspects. We could derive an upper bound on the worst case performance. In the worst case, rules lead to splitting and storing simplices that have a size slightly larger than α. After

Algorithm 3 : Evaluate a point and update global information.

```
Funct EvaluateVertex ($v$); global $\Lambda$, $f^U$, $x^U$
1. Determine $f(v) = c^T v$ and $g_i(v)$, $i = 1, \dots, m$
2. Check whether $v \in X \cap D$                     Evaluate linear and quadratic feasibility of v
3. if $f(v) < f^U$
4.   if $v \in X$ and is $\epsilon$-robust                                  See Sect. 4
5.       $f^U := f(v)$ and $x^U := v$                            Update global information
6.       Remove all $C_k \in \Lambda$ such that $f_k^L > f^U$                  Cutoff test
```

going $n(n-1)/2$ levels deeper in the search tree, at least all edges have been halved and the size of the simplex is less or equal than half its original size. Suppose $Size(C_1) = 1$ (infinity norm distance). The maximum number K of halving the simplices is given by $1/2^K \leq \alpha$, such that $K = \lceil(-\ln(\alpha)/\ln(2))\rceil$, where $\lceil y \rceil$ is the lowest integer greater than or equal to y. Given the number of edges per simplex $n(n-1)/2$, the maximum depth of the search tree is $K \times n(n-1)/2$. The final level is not stored, as the simplices do not pass the size test. An overestimate for the worst case number of simplices on the list is:

$$2^{K \times n(n-1)/2-1}, \tag{10}$$

where $K = \lceil(-\ln(\alpha)/\ln(2))\rceil$. This analysis provides a guarantee that the algorithm is finite given an accuracy α. We will observe in the experiments that practically the number of simplices the algorithm generates and evaluates is much lower. The real success of Branch-and-Bound depends on how early branches of the tree can be pruned.

3 Infeasibility check

After verifying that all vertices $v \in C$ are infeasible, one would like to test whether C is completely infeasible. In [6], one constructs a lower bound $g^L \leq g(x), x \in C$ on C for quadratic function $g(x) = x^T Ax + b^T x + d$, such that one can delete C with certainty when $g^L > 0$. This "bounding by infeasibility" step requires the determination of a lower bound of a quadratic function on a simplicial set. As noted by [1,2], finding the exact minimum is a hard problem (if not convex) and lower bounds can be constructed in several ways. Actually for the infeasibility test neither the exact minimum, nor a lower bound is necessary.

Given vertex v with value $g(v) > 0$, one would like to know when taking a deviation step r from v into the simplex that the value of g changes so much that it can drop below zero. For a quadratic function the absolute change in value can be written as

$$\begin{aligned} |g(v+r) - g(v)| &= |(v+r)^T A(v+r) + b^T r - v^T Av| \\ &= |r^T(2A(v + \frac{1}{2}r) + b)| = |r^T \nabla g(v + \frac{1}{2}r)|. \end{aligned} \tag{11}$$

There are several ways to overestimate the absolute change. For $y \in C$ the Lipschitzian view gives

$$|r^T \nabla g(y)| \leq \|r\| \times \|\nabla g(y)\| \leq Size(C) \times L_2(C), \tag{12}$$

where $L_2(C) = \max_{y \in C} \|\nabla g(y)\|_2$ is a Euclidian Lipschitz constant over C and $Size(C)$ the longest edge according to the Euclidian norm. An alternative is to switch norm according to

$$|r^T \nabla g(y)| = \left| \sum_j r_j \frac{\partial g}{\partial x_j}(y) \right| \leq \sum_j |r_j| \times \left| \frac{\partial g}{\partial x_j}(y) \right|$$
$$\leq \max_j |r_j| \times \sum_j \left| \frac{\partial g}{\partial x_j}(y) \right| \leq Size_\infty(C) \times L_\infty(C), \tag{13}$$

where $L_\infty(C) = \max_{y \in C} \|\nabla g(y)\|_1$ and $Size_\infty(C)$ is the longest edge according to the infinity norm. The maximum gradient length (L_2 as well as L_∞) can be determined by considering the values in the vertices, as it implies maximising a convex function over a simplex.

This estimation can be made sharper, by considering in (12) that in fact r moves into the plane of the unit simplex. This means, one can take for the Lipschitz constant the maximum norm over the partial derivatives that are perpendicular to the all ones vector $\mathbf{1}$. To say it in another way, instead of taking for $L(C)$ the maximum norm of $\nabla g(y)$, one takes the norm of the projection p of $\nabla g(y)$ on the unit simplex. Let $\gamma = \nabla g(y)$, then

$$p = \gamma - \frac{1}{n}\mathbf{1}^T \gamma \mathbf{1}. \tag{14}$$

This means that instead of basing the Lipschitz constant on the Euclidian length $\sqrt{\sum \gamma_j^2}$ of $\gamma = \nabla g(y)$, we can base it on the maximum norm of the projected directional derivatives p, being $\sqrt{\sum \gamma_j^2 - \frac{1}{n}(\sum \gamma_j)^2}$, which is obviously smaller.

Finally, the feasibility check reduces to labeling a simplex C infeasible with respect to g if

$$\max_{v \in vertices(C)} g(v) - L(C) \times Size(C) > 0. \tag{15}$$

Equation 15 can be made sharper to achieve lower computational costs by considering

$$\max_{v \in vertices(C)} \left(g(v) - L(C) \times \max_{x \in C} \|v - x\| \right) > 0. \tag{16}$$

Both equations have the typical Lipschitzian view that can also be found in attempts to apply the Lipschitz constant in higher dimensions. Mladineo [13] derives a lower bound by equating over x the cones φ_j defined by:

$$\varphi_j(x) = g(v_j) - L\|x - v_j\|, \tag{17}$$

for the $n + 1$ vertices v_j of a simplicial region C in $\mathbb{R}^n$. This leads to a set of equalities that is not easy to solve. In [11] and [14] approximations of (17) are introduced over rectangular regions for deriving lower bounds. More recently (in [3]) one can find results of applying the underestimation via (15) and (16) on simplicial sets.

Consider again the grid search benchmark. One of the ideas to generate a grid over the simplex is, that in some cases one can say with certainty that a feasible solution does not exist. A sufficient but not necessary condition is given in the following theorem.

Theorem 1 *Given unit simplex S, a regular grid defined by grid points v_j with M points per axis, g_i quadratic functions with Lipschitz constants Γ_i, $i = 1, \ldots, m$ and the problem of finding solutions of D. If*

$$\max_i \frac{g_i(v_j)}{\Gamma_i} > \frac{\sqrt{2}}{M - 1} \quad \forall j, \tag{18}$$

then S does not contain any feasible solution of D ($S \cap D = \emptyset$).

Proof Given the grid over S

$$\forall x \in S \ \exists j \ \ \|x - v_j\| \leq \frac{\sqrt{2}}{M-1}. \tag{19}$$

Combining (18) and (19) gives

$$\forall x \in S \ \exists i, j \ \ g_i(x) \geq g_i(v_j) - \Gamma_i \|x - v_j\| \geq g_i(v_j) - \Gamma_i \frac{\sqrt{2}}{M-1} > 0. \tag{20}$$

□

Equations 15 or 16 try to answer the question whether $g(x)$ can reach a value lower than zero on C if all of its vertices are infeasible with respect to g, i.e., $g(v_j) > 0$ for all vertices v_j. If for one of the vertices $g(v_j) \leq 0$, the lower bounding question will not help to discard the simplex. The infeasibility question can be made sharper by considering all requirements $g_i(v) \leq 0$ simultaneously for $i = 1, \ldots, m$, instead of considering the requirements individually. Let us consider the case when all vertices are infeasible for at least one of the functions, $g_i(v_j) > 0$. Around each vertex v_j, a so-called infeasibility sphere B_j is defined that cannot contain a feasible point

$$B_j = \{x \in \mathbb{R}^n, \|x - v_j\|^2 < \rho_j^2\}. \tag{21}$$

Two possible ways of calculating a value for ρ_j will be discussed. First, based on constants:

$$\Gamma_i(C) = \max_{v \in C} \left\| \nabla g_i(v) - \frac{\mathbf{1}^T \nabla g_i(v) \mathbf{1}}{n} \right\|, \tag{22}$$

one can derive infeasibility radii

$$\rho_j^l(C) = \max_i \frac{g_i(v_j)}{\Gamma_i(C)}. \tag{23}$$

Notice that $\rho = \max_j \rho_j^l > \max_{x \in C} \|v_j - x\|$ means that C is completely infeasible corresponding to Eq. 16.

An infeasibility radius can also be derived from quadratic considerations based on the following theorem.

Theorem 2 *Given a quadratic function $g(x) = x^T Ax + b^T x + d$ with a symmetric n by n matrix A with the most negative eigenvalue η, b an n-vector and d a scalar. If $g(v) > 0$ then $g(v + r) > 0$ for $\|r\| < \frac{\|\nabla g(v)\| - \sqrt{\|\nabla g(v)\|^2 - 4\eta g(v)}}{2\eta}$.*

Proof

$$\begin{aligned} g(v + r) &= (v + r)^T A(v + r) + b^T (v + r) + d \\ &= g(v) + r^T (2Av + b) + r^T Ar \\ &\geq g(v) - \|r\| \|\nabla g(v)\| + \eta \|r\|^2 \end{aligned} \tag{24}$$

Solving over ρ

$$g(v) - \rho \|\nabla g(v)\| + \eta \rho^2 = 0 \tag{25}$$

gives as positive root $\rho = \frac{\|\nabla g(v)\| - \sqrt{\|\nabla g(v)\|^2 - 4\eta g(v)}}{2\eta}$.
Given $g(v) > 0$ leads to $g(v + r) > 0$ in (24) for $\|r\| < \rho$. □

To derive infeasibility spheres using Theorem 2, we have to determine the lowest eigenvalues $\eta_1, \ldots, \eta_m$ of $A_1, \ldots, A_m$. Notice that if the function g_i is convex, the smallest eigenvalue is positive and Theorem 2 does not apply. The quadratic infeasibility radii are given by:

$$\rho_j^q = \max_{i \ with \ \eta_i < 0} \frac{\|\nabla g_i(v_j)\| - \sqrt{\|\nabla g_i(v_j)\|^2 - 4\eta_i g_i(v_j)}}{2\eta_i}. \tag{26}$$

For more negative curvature, i.e., η_i is more negative, the radius is smaller. Therefore less negative values of η are better. Similar to the projection of the gradient, one can improve its value because the deviating point $x + r$, or its projection, is moving over the unit simplex. This means, it is sufficient to consider the most negative eigenvalue of matrix $P^T AP$, when the projection matrix P is given by

$$P = \frac{1}{n}\begin{pmatrix} n-1 & -1 & . & . & -1 \\ -1 & n-1 & \ddots & . & . \\ . & \ddots & \ddots & \ddots & . \\ . & . & \ddots & n-1 & -1 \\ -1 & . & . & -1 & n-1 \end{pmatrix}. \tag{27}$$

Finally, to get the largest infeasibility sphere at vertex j, we take:

$$\rho_j = \max\{\rho_j^l(C), \rho_j^q\}. \tag{28}$$

The idea of infeasibility spheres handles well the concept of considering several constraint functions at the same time. To go one step further, we can also consider the infeasibility spheres B_j of all vertices simultaneously. For the Branch-and-Bound algorithm, the question can be shifted to covering by what we call the raspberry set $Rasp(C) = \cup_j B_j$ (see [5]). In the algorithm, the (in)feasibility check can be done by answering the question: does $Rasp(C)$ cover set C, i.e., is $C \subset Rasp(C)$? It is not straightforward to answer this question. To check this, one could look for a point in $C \setminus Rasp(C)$. However, the following theorem, illustrated by Fig. 3, shows that it is sufficient to look—the other way around—for a point that is covered by all infeasibility spheres.

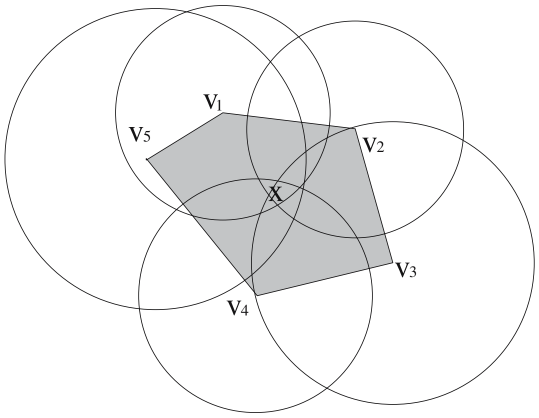

Fig. 3 Polytope covered by raspberry set

Theorem 3 *Given a set* $V = conv\{v_1, \ldots, v_h\}$, *with* $v_1, \ldots, v_h$ *the extreme points of* V (*vertices*). *Let each vertex* v_j *be infeasible, i.e.,* $\max_i g_i(v_j) > 0$ *with corresponding infeasibility sphere* B_j *defined by* (21). *If* $\exists x \in \cap_j B_j \cap V$ *then* $V \subset \cup_j B_j$ *and consequently no feasible point exists in* V.

Proof Assume that $\exists y \in V$ such that $y \notin \cup_j B_j$. We will show that this assumption leads to a contradiction. Define the vector $r = y - x$ as the direction of stepping from point x to the point y that is assumed not to be covered. Moreover, we will consider point $z = \frac{x+y}{2} = x + \frac{1}{2}r$ that is in the middle of x and y. From $x \in \cap_j B_j$ and $y \notin \cup B_j$ follows that $\forall j \ \ \|y - v_j\|^2 \geq \rho_j^2$ and $\|x - v_j\|^2 < \rho_j^2$. Walking from one point to the other gives a mean-value result:

$$\begin{aligned}\|y - v_j\|^2 &= (x + r - v_j)^T(x + r - v_j)\\ &= (x - v_j)^T(x - v_j) + 2r^T(x - v_j) + r^T r\\ &= \|x - v_j\|^2 + 2r^T(x + \frac{1}{2}r - v_j) < \rho_j^2 + 2r^T(z - v_j).\end{aligned} \tag{29}$$

Combining $\|y - v_j\|^2 \geq \rho_j^2$ with (29) leads to the conclusion that

$$2r^T(z - v_j) > 0, \quad j = 1, \ldots, h. \tag{30}$$

Equation 30 tells us that the assumption of the existence of y leads to the appearance of a point $z \in V$ and a direction r such that the directional derivative with respect to the squared distance function is positive with respect to all vertices $v_1, \ldots, v_h$. This means that walking from z in direction r makes us go further away from all vertices simultaneously. This is in contradiction with z being in V.

More exactly, function $f(a) = r^T a$ is linear in a and therefore attains its maximum over polytope V in one of the vertices v_p. For that vertex, $\max_{a \in V} 2r^T(a - v_p) = 0$ such that Eq. 30 cannot be true for at least one vertex v_j. This means that a point y with $y \notin B_j \forall j$ cannot exist in set V. □

Theorem 3 shows that in order to prove infeasibility of C, it is sufficient to find a point $x \in C$ that for each vertex v_j is closer than ρ_j. One can make a good guess by taking a weighted average

$$\xi = \frac{1}{\sum_j \frac{1}{\rho_j}} \sum_j \frac{v_j}{\rho_j} \tag{31}$$

and test whether $\|\xi - v_j\|^2 < \rho_j^2$ for each vertex v_j. If ξ is not covered by the smallest sphere, i.e., $\|\xi - v_k\|^2 \geq \rho_k^2$ with $k = \arg\min_j \rho_j$, one would like to suggest another trial point. It can be obtained heuristically by walking in the direction of v_k and generating a point θ that is at least covered by the smallest sphere:

$$\theta = v_k + (\rho_k - \delta)\frac{\xi - v_k}{\|\xi - v_k\|}, \tag{32}$$

where δ is a small positive accuracy number, such that θ is just within the interior.

The theory is used to derive the following tests for proving infeasibility of subset C:

– The LITest (Linear Infeasibility Test) consists of eliminating a simplex C in case for one of the linear constraints all its vertices are infeasible.

- The SCTest (Single sphere Cover Test) proves that a simplex is infeasible because it is completely covered by an infeasibility sphere. So the SCTest consists of determining a vertex v_k satisfying $\rho_k > \max_{x\in C} \|v_k - x\|$.
- The NCTest (N spheres Cover Test) tries to determine if a simplex C is completely covered by the set of all infeasibility spheres B_j; i.e., $C \subset \cup_j B_j$. It is basically the practical application of Theorem 3 using the point defined by Eqs. 31 and 32.
- The PCTest is applied when the SCTest and NCTest fail. Either ξ or θ (if infeasible) is used to generate an infeasibility sphere centered in it which is then checked to cover C.
- A simplex with an infeasible point x such that $\max_j \|x - v_j\| < \epsilon$, cannot contain an ϵ-robust solution. The ϵ-infeasibility test checks this condition for all vertices and the point ξ or θ.

4 ϵ-robustness determination

Robustness $R(x)$ of a design x with respect to set D has been introduced in Eq. 7. For quadratic inequalities it is not easy to determine the nearest infeasible point to x, such that algorithms as described in [4] are needed. We are merely interested in generating ϵ-robust solutions. Although the algorithm does not guarantee to find such points, one can generate a valid lower bound $R^L(x)$ on the robustness. If the lower bound $R^L(x)$ is larger than ϵ, point x is ϵ-robust. Similar to (15) one can define for a feasible point x with $g_i(x) < 0, i = 1, \ldots, m$

$$R^L_\Gamma(x) = \min_i \frac{-g_i(x)}{\Gamma_i}. \tag{33}$$

As long as $\|h\|$ is smaller than $R^L_\Gamma(x)$ then $g_i(x+h) \le g_i(x) + \Gamma_i\|h\| \le 0$ and point $x+h$ fulfills all constraints. For Γ_i one can take the maximum Euclidian length of the projected gradient over the unit simplex S.

Alternatively, in the line of Theorem 2, one can base a lower bound on the quadratic shape of the function

$$\begin{aligned} g(x+h) &= (x+h)^T A(x+h) + b^T(x+h) + d \\ &= g(x) + h^T(2Ax+b) + h^T Ah \\ &\le g(x) + \|h\|\|\nabla g(x)\| + \mu\|h\|^2, \end{aligned} \tag{34}$$

where μ is the largest eigenvalue of A. Solving for maximum step size $\|h\|$ equating the overestimation (34) to 0, gives a lower bound R^L_Q for the maximum step size of $\|h\|$ such that $x+h$ is still feasible with respect to the quadratic constraint. To be more specific, we determine the maximum eigenvalues μ_i of A_i and compute the lower bound robustness based on quadratic considerations as

$$R^L_Q(x) = \min_{i:\, \mu_i>0} \frac{-\|\nabla g_i(x)\| + \sqrt{\|\nabla g_i(x)\|^2 - 4\mu g_i(x)}}{2\mu}. \tag{35}$$

This shows more clearly that if second derivatives are large, the robustness estimation is worse. Therefore smaller values of μ are better. Similar to the projection of the gradient, one can slightly improve its value considering that point $x+h$ (or its projection) is moving over the unit simplex. This means, it is sufficient to consider the largest eigenvalue of matrix $P^T AP$, where the projection matrix P is given by Eq. 27. The lower bound on the robustness $R^L(x)$ is taken as $\max\{R^L_\Gamma, R^L_Q\}$ to be confronted with the ϵ-robustness that one intends to reach.

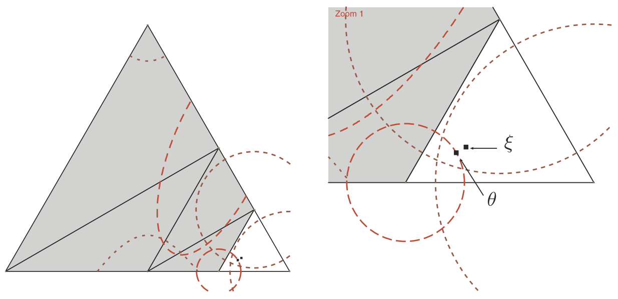

Fig. 4 One of the iterations of the B&B algorithm for RumCoke. The graph at the right is a zoom of the graph at the left in the figure

The solution provided as output of Algorithm 1 (if any) is a point x such that $x \in X \cap D$ meets the condition $R^L(x) \geq \epsilon$ with the lowest cost found.

5 Illustrative examples

The algorithm was used in a larger context of the practical product design project, where application to larger instances saved hours of calculation time (see [8]). For the illustration, two 3-dimensional cases were modified from [6]. The data are given in Appendix 2. The advantage of the three-dimensional cases is that the algorithm execution can be graphically represented. The first case, called RumCoke (RC), is an example to illustrate mixture design with two quadratic constraints. The second case, Case-2 (C2), was taken from an industrial example having five quadratic constraints. The algorithm was also run on larger cases from industry, where we took one of the cases to illustrate the contribution of the developed theory. A standard test set for performance comparison of cases for problem (8) does not exist yet in the literature.

Figure 4 shows one of the iterations of the algorithm for RumCoke. Each requirement has been represented by a different dashed line (and colour). Each infeasible sphere (21) has been represented by the same dashed (and coloured) line as its corresponding quadratic constraint. The simplices currently on list Λ are given in grey. Focusing on the white simplex that is currently evaluated, one can observe that none of the spheres individually covers the simplex completely. Weighted average ξ is not covered by all spheres, but modified point θ, close to the boundary of the smallest sphere, is covered, so this simplex is eliminated by the NCTest.

The next step is to have the B&B results comparable with evaluating the points on a regular grid. For the illustration, we will use the bisection process until every axis is bisected K times. Values of K used in the illustration are: $K = 8$, $K = 9$, and $K = 10$, such that the corresponding mesh size is $(\frac{1}{2})^K$ ($\frac{1}{256}$, $\frac{1}{512}$, and $\frac{1}{1024}$), implying $M = 2^K + 1$ points per axis ($M = 257$, $M = 513$, and $M = 1025$). This means that the corresponding grids contain 33153, 131841 and 525825 vertices, respectively. In the algorithm, to obtain the depth of K, one should—working with Euclidian norm—apply an accuracy α somewhere in between $\frac{\sqrt{2}}{2^{K-1}}$ and $\frac{\sqrt{2}}{2^K}$. Table 1 shows results for a robustness of $\epsilon = 0.01$ when the accuracy α is varied. For all the values of α tested in our experiments the algorithm has found a robust solution.

The Branch-and-Bound algorithm evaluates points and stores vertices (points) and simplices that need to be explored further. Table 1 gives the value of the cost function of the

Table 1 Regular mesh vs. Branch-and-Bound algorithm $\epsilon = 0.01$, $(n = 3)$

Regular Grid				Branch-and-Bound					
K	M	Vertex	α	Vertex		Simplex		Cost	
				RC	C2	RC	C2	RC	C2
8	257	33,153	0.0075	170	221	389	499	0.6104	1.3965
9	513	131,841	0.0050	161	493	373	220	0.5916	1.3813
10	1,025	525,825	0.0020	199	254	465	577	0.5753	1.3691

Table 2 Efficiency of Theorem 3, $\epsilon = 0.005$

Problem	B&B with Theorem 3			Without Theorem 3		
	RumCoke	Case2	UniSpec	RumCoke	Case2	UniSpec
NSimplex	373	493	$7.9\ 10^6$	403	599	$8.9\ 10^6$
NVertex	183	241	$2.6\ 10^6$	199	293	$2.9\ 10^6$
ϵ-infeas.	52	88		54	100	
Cutoff	45	17	$326\ 10^3$	47	19	$356\ 10^3$
LITest	3		$226\ 10^3$	3		$279\ 10^3$
SCTest	42	74	$1{,}022\ 10^3$	64	143	$1{,}747\ 10^3$
Thrm 3	11	30	$545\ 10^3$			

best robust design found (column Cost) and the number of evaluated points (column Vertex) and simplices (column Simplex) to reach the result for problems RumCoke (column RC) and Case 2 (column C2). It can also be seen that increasing the value of the accuracy, the algorithm is able to obtain solutions with better cost function values. The efficiency of the B&B algorithm looks promising. For both problems the number of vertices evaluated by the B&B algorithm is between 340 and 2,300 times less than the number of vertices on the corresponding regular grid. Additional computational effort is necessary to evaluate the set of generated simplices. Once an ϵ-robust solution has been found, the cutoff test with respect to cost starts working.

A next task is to measure the effect of the introduced new theoretical results. Actually, the idea of infeasibility spheres is new as such, but close to earlier introduced conical ideas based on Lipschitz constants. The question is now what Theorem 3 adds to the algorithm. It introduces the N spheres Cover Test and generates interior points θ and ξ. These points are evaluated and possibly used again to perform an ϵ infeasibility test (are all points of the simplex closer than ϵ to the point?) or alternatively to construct an infeasibility sphere (the PCTest). What happens to the algorithm when we leave out these tests derived from Theorem 3? To measure this, the algorithm is run with and without the corresponding tests and the efficiency in number of evaluated and rejected simplices and vertices is measured. The result can be observed in Table 2.

The cases used, are the illustrative cases RumCoke and Case 2 and a seven-dimensional industrial problem taken from Unilever Research called UniSpec. The number of evaluated simplices (NSimplex) and vertices (NVertex) is measured. The disposed simplices due to ϵ-infeasibility with respect to vertices, objective function bounding (Cutoff), Single Cover Test, Linear infeasibility and the tests due to Theorem 3 (Thrm 3) is noted for the three cases using a robustness $\epsilon = 0.005$. For the three-dimensional cases, an accuracy of $\alpha = 0.005$ was taken. One can observe for these cases, where the total number of evaluations is low, that Theorem 3 leads to a reduction of evaluations of some 5–20%. In total, less subsets are being rejected by the other tests, because part of the search space is discarded by the N spheres Cover Test. The larger dimensional case was run with an accuracy $\alpha = 0.05$. Millions of subsets are generated and vertices are evaluated corresponding to megabytes of storage requirements. The savings due to application of the results of Theorem 3, are of the same order of magnitude as for the lower dimensional cases.

6 Conclusions

This paper shows that a Branch-and-Bound algorithm based on bisection over the longest edge with an accuracy stopping rule, in our case on size α, generates points on a regular grid. From this viewpoint, a Branch-and-Bound algorithm tries not to evaluate all grid points by removing areas where the optimal solution is proven not to be located. Theoretical results on generating infeasibility spheres have been presented. A new result on how to combine these spheres to prove infeasibility of a polytope has been discussed. Moreover, the results have been implemented in practical computational tests.

Based on these tests, a Branch-and-Bound algorithm to find robust solutions of a mixture design problem has been presented. The algorithm is able to detect cases that do not contain ϵ-feasible solutions. The new aspect of the algorithm is that it guarantees that the generated solutions, if any, are ϵ-robust. More specifically, the algorithm can return three different kinds of results.

- The output of the algorithm is an approximation of the best ϵ-robust solution. The approximation becomes better if the accuracy α goes to zero.
- The algorithm returns: "No solution exists". The algorithm has proven that no ϵ-feasible solution exists.
- The third possible output of the algorithm is: "No solution found", which means no robust solution has been found, but in theory it may exist.

The whole has been illustrated with low dimensional design problems. Applying the ideas presented here in a larger project, the number of points evaluated by the algorithm and the storage requirements for generating the same result have been reduced considerably.

Acknowledgements This work has been partially supported by the Ministry of Education and Science of Spain through grant TIN2005-00447.

Appendix 1

A regular grid with accuracy $\alpha = \frac{1}{M-1}$ can be generated by the recursive procedure $GiveValue(M, n)$ over the unit simplex, where M is the number of grid points per axis. The resulting number of points is very high for a reasonable accuracy. This is illustrated in Table 3, where the number of points on the regular grid is given for $M = 11$ and $M = 101$. The number of points results as well from Algorithm 4 as from Eq. 9.

Table 3 Number of regular grid points on the unit simplex

M	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$
11	11	66	286	1,001	3,003	8,008
101	101	5151	176,851	$4.6\,10^6$	$9.7\,10^7$	$1.7\,10^9$

Algorithm 4 : Grid on a simplex

```
Funct GiveValue ($P, m$)
1. for $k = 0$ to $(P - 1)$
2.     $x_m = \alpha k$
3.     if $m \geq 3$
4.         GiveValue($P - k, m - 1$)
5.     $x_1 = 1 - \sum_{j=2}^{n} x_j$
6. return $x_1, \ldots, x_n$
```

Appendix 2: Illustrative cases

RumCoke

Dimension = 3; Raw material cost = {0.1, 0.7, 4.0}
Linear Constraints:

$$h_1(x) = -1.5x_1 + 0.5x_2 - 0.5x_3 \geq 0.0$$
$$h_2(x) = 0.3x_1 - 0.5x_2 - 0.3x_3 \geq 0.0$$

Quadratic constraints ($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0 \;\; i = 1, 2$):

$$A_1[3 \times 3] = \{0, -16, 0, -16, 0, 0, 0, 0, 0\}$$
$$b_1[3 \times 1] = \{8, 8, 0\};\, d_1 = -1$$
$$A_2[3 \times 3] = \{10, 0, 2, 0, 0, 0, 2, 0, 2\}$$
$$b_2[3 \times 1] = \{-12, 0, -4\};\, d_2 = 3.7$$

Case2

Dimension = 3; Raw material cost = {1.1,1.7,2.0}
Quadratic constraints ($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0 \;\; i = 1, \ldots, 5$):

$$A_1[3 \times 3] = \{0.001, -0.001, 0.0085, -0.001, 0.008, -0.0105, 0.0085, -0.0105, -0.021\}$$
$$b_1[3 \times 1] = \{-0.0145, -0.0205, 0.073\};\, d_1 = -0.0165$$
$$A_2[3 \times 3] = \{-0.004, 0.0005, 0.002, 0.0005, -0.001, -0.003, 0.002, -0.003, 0.014\}$$
$$b_2[3 \times 1] = \{0.0155, 0.0515, -0.121\};\, d_2 = -0.006$$
$$A_3[3 \times 3] = \{20.605, -5.087, -10.9885, -5.087, 32.003, -43.476, -10.9885, -43.476, -81.278\}$$
$$b_3[3 \times 1] = \{0.1995, -0.097, 126.7685\};\, d_3 = -20.5063$$
$$A_4[3 \times 3] = \{0.766, -0.1205, 2.4735, -0.1205, 0.528, 1.9835, 2.4735, 1.9835, -7.822\}$$
$$b_4[3 \times 1] = \{-2.432, -15.191, 10.712\};\, d_4 = 3.21125$$

$A_5[3 \times 3] = \{116.75, -3.09, 168.553, -3.09, -67.424, 515.114, 168.553, 515.114, -845.215\}$

$b_5[3 \times 1] = \{-287.43, -645.926, 354.537\}; d_5 = 115.0953$

UniSpec
Dimension = 7; Raw material cost = (114, 115, 107, 127, 115, 106, 108)
Linear Contraint:

$$h_1(x) = 0.1493x_1 + 0.6927x_2 + 0.4643x_3 + 0.7975x_4 + 0.5967x_5 + 0.6235x_6 + 0.5284x_7 \geq 0.35$$

Quadratic constraints ($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 1, 2, 3$):

$A_1[7 \times 7] = (-1.473, 8.215, -27.204, 46.119, 2.059, -11.929, -12.768, 8.215, 37.733346, 5.127, 95.691, 34.954, 20.165, 19.445, -27.204, 5.127, -21.743, 36.843, -7.126, 4.029, -4.152, 46.119, 95.691, 36.843, 189.643, 93.359, 52.904, 54.802, 2.059, 34.954, -7.126, 93.356, 31.885, 7.528, 10.248, -11.929, 20.165, 4.029, 52.904, 7.528, 11.951, 10.964, -12.768, 19.445, -4.152, 54.802, 10.248, 10.964, 7.197)$

$b_1[7 \times 1] = (4.5675, 34.7289, 70.5707, -82.2761, 29.3169, 71.0818, 63.7614);$

$d_1 = -35$

$A_2[7 \times 7] = (1.35, -4.41, 17.60, -92.45, 2.74, -29.94, -14.05, -4.41, -39.13, -6.11, -126.38, -29.81, -63.42, -43.97, 17.60, -6.11, 15.45, -76.60, 5.93, -44.05, -20.54, -92.45, -126.38, -76.60, -240.64, -117.46, -125.18, -114.98, 2.74, -29.81, 5.93, -117.46, -22.90, -47.37, -30.68, -29.94, -63.42, -44.05, -125.18, -47.37, -73.39, -73.99, -14.05, -43.97, -20.54, -114.98, -30.68, -73.99, -55.33)$

$b_2[7 \times 1] = (-2.1232, -9.0403, -42.2072, 190.5292, -9.9529, 1.8162, 5.1622);$

$d_2 = 10$

$A_3[7 \times 7] = (-0.670, 4.284, -12.837, 23.708, 1.677, -8.964, -4.859, 4.284, 21.380, -1.188, 28.990, 13.216, 17.177, 16.620, -12.837, -1.189, -21.376, 9.841, -7.298, -10.043, -8.981, 23.708, 28.990, 9.841, 49.385, 25.574, 15.561, 21.666, 1.677, 13.216, -7.298, 25.574, 8.419, 4.149, 6.595, -8.965, 17.177, -10.043, 15.561, 4.149, 1.090, 6.292, -4.859, 16.620, -8.981, 21.666, 6.594, 6.292, 5.906)$

$b_3[7 \times 1] = (0.7097, -13.0982, 27.5078, -49.1608, -7.3725, 33.6731, 11.3136);$

$d_3 = -2$

References

1. Anstreicher, K.M., Burer, S.: D.C. versus copositive bounds for standard QP. J. Global Optim. **33**, 299–312 (2005)
2. Bomze, I.M., de Klerk, E.: Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. J. Global Optim. **24**, 163–185 (2002)

3. Claussen, J., Zilinskas, A.: Subdivision, sampling and initialization strategies for simplical branch and bound in global optimization. Comput. Math. Appl. **44**, 943–955 (2002)
4. Hendrix, E.M.T.: Global Optimization at Work. Wageningen University, Wageningen (1998)
5. Hendrix, E.M.T., Klepper, O.: On uniform covering, adaptive random search and raspberries. J. Global Optim. **18**(2), 143–163 (2000)
6. Hendrix, E.M.T., Pintér, J.D.: An application of Lipschitzian global optimization to product design. J. Global Optim. **1**, 389–401 (1991)
7. Hendrix, E.M.T., Mecking, C.J., Hendriks, T.H.B.: Finding robust solutions for product design problems. Eur. J. Oper. Res. **92**, 28–36 (1996)
8. Hendrix, E.M.T., Casado, L.G., García, I.: The semi-continous quadratic mixture design problem. Description and branch-and-bound approach. In press, doi:10.1016/j.ejor.2007.01.056 (2007). Eur. J. Oper. Res.
9. Horst, R.: On generalized bisection of n-simplices. Math. Comput. **66**(218), 691–698 (1997)
10. Ibaraki, T.: Theoretical comparisons of search strategies in branch and bound algorithms. Int. J. Comput. Inform. Sci. **5**, 315–344 (1976)
11. Meewella, C.C., Mayne, D.Q.: An algorithm for global optimization of Lipschitz continuous functions. J. Optim. Theory Appl. **57**, 307–322 (1988)
12. Mitten, L.G.: Branch and bound methods: general formulation and properties. Opera. Res. **18**, 24–34 (1970)
13. Mladineo, R.H.: An algorithm for finding the global maximum of a multimodal, multivariate function. Math. Program. **34**, 188–200 (1986)
14. Pintér, J.D.: Branch-and-bound algorithms for solving global optimization problems with Lipschitzian structure. Optimization **19**, 101–110 (1988)
15. Williams, H.P.: Model Building in Mathematical Programming. Wiley & Sons, Chichester (1993)