WORKING PAPER
MANSHOLT GRADUATE SCHOOL

On Blending with Lipschitzian requirements

Eligius M.T. Hendrix, L.G. Casado and I. García

DISCUSSION PAPER No.16
2005

Mansholt Graduate School

Hollandseweg 1, 6706 KN Wageningen,
The Netherlands
Phone: +31 317 48 41 26
Fax: +31 317 48 47 63
Internet: http://www.mansholt.wur.nl/
e-mail: office.mansholt@wur.nl

Eligius M.T. Hendrix[a]

[a]Operations Research en Logistics Group, Wageningen University, Hollandseweg 1, 6706 KN Wageningen, The Netherlands.

[1] Working papers may have been submitted to other journals and have entered a journal's review process. Should the journal decide to publish the article the paper no longer will have the status of a Mansholt Working Paper and will be withdrawn from the Mansholt Graduate School's website. From then on a link will be made to the journal in question referring to the published work and its proper citation.

On Blending with Lipschitzian requirements[*]

Eligius M.T. Hendrix, L.G. Casado and I. García

Operationele Research en Logistiek Groep, Wageningen Universiteit.

Dpt. de Arquitectura de Computadores y Electronica, Universidad de Almería.

May 5, 2005

Abstract

The blending problem is studied as a problem of finding feasible solutions on the unit simplex fulfilling Lipschitzian inequalities. It is sketched how such problems can be solved by Branch-and-Bound type of algorithms. A new infeasibility test is derived that can be used in Branch-and-Bound approaches. Some properties of a regular grid over the unit simplex are discussed. The whole is illustrated with some numerical examples.

keywords: Blending, Branch-and-Bound, Lipschitz continuity.

1 Introduction

The blending problem is usually posed as finding a mixture represented by a point on the unit simplex $S = \{x \in R^n | \sum_j x_j = 1.0;\ x_j \geq 0\}$ that is as cheap as possible and also fulfills quality requirements. The variables x_j represent the fraction of the components in a product x. Such problems are solved on a daily base in fodder and petrochemical industry where often requirements are modelled by linear inequalities, see e.g. [12]. 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. play a role. Here, we focus on that aspect of the design process, where requirements have a nonlinear, Lipschitz continuous structure. This means that the mathematical problem we are interested in is posed by finding feasible solutions on the unit simplex of a set of inequalities

$$g_i(x) \leq 0; \quad i = 1, \ldots, m \tag{1}$$

[*]This work has been partially supported by the Ministry of Education and Science of Spain through grant CICYT-TIC2002-00228

where the functions $g_i(x)$ are Lipschitz continuous on S; i.e. for each function g_i there exists a Lipschitz constant L_i such that

$$| \ g_i(x) - g_i(y) \ | \ \leq \ L_i \|x - y\|, \ \ \forall x, y \in S \tag{2}$$

In the practical setting the problem originates from, the search for feasible solutions was approached by running standard solvers on a penalty reformulation:

$$\min_{x \in S} \{ \max_i g_i(x) \} \tag{3}$$

or

$$\min_{x \in S} \{ \sum_i \max[g_i(x), 0] \} \tag{4}$$

The challenging aspect is that optimizing (3) or (4) is a global optimization problem. This means that applying several starting points may result into local optima that are all bigger than zero, i.e., no feasible solution of (1) is attained. The question is: How to be certain that no feasible solution exists? Seen from the viewpoint of mathematical programming, one can make use of the Lipschitzian structure by for instance constructing a specific algorithm based on Branch-and-Bound, see e.g. [5]. Early types of specific Lipschitzian algorithms for optimization of functions in one dimension are due to [11] and [1]. Extension for functions in more than one dimension followed some time after, e.g. [7], [9] and [10]. Practical use of such ideas for industrial purposes can be found in [4], where properties g_i are quadratic functions, but handled from a Lipschitz perspective.

In this article, a new theoretical result is presented on how to determine that a polytope cannot contain a feasible point of (1). This result is useful in the context of a Branch-and-Bound algorithm. Therefore in Section 2 an algorithm is sketched to determine whether a solution of (1) exists. In Section 3, theoretical results are presented. Moreover, the efficiency of the method of grid search is analysed. Numerical experiments with cases derived from industry are shown in Section 4. Finally conclusions are drawn in Section 5.

2 Algorithms for finding a solution

One approach, as sketched before, is to run nonlinear optimization routines on penalty formulation (3) or (4). However, if the final answer is greater than zero, the statement "no solution exists" cannot be given with absolute certainty. Guarantee with a certain accuracy can be given, if one would generate a regular grid over the unit simplex, evaluating all the points as sketched in Figure 1.

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 very 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

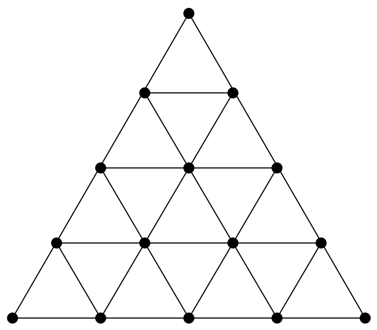

Figure 1: 2-dim projection of regular grid over the unit simplex, M=5.

$\sum_j x_j = 1.0$. This is illustrated in Figure 1. It can be verified that the total number of points on the grid is given by

$$\sum_{k=1}^{n-1} \binom{M}{k} \binom{n-2}{k-1} \tag{5}$$

as sketched in the appendix. This means that the number of points, although very high, is not exponential in the dimension. For the example in Figure 1, $n = 3$ and $M = 5$, we have 15 grid points and an "accuracy" of $\alpha = 0.25$.

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 a solution. B&B methods can be characterized by four rules: *Branching, Selection, Bounding*, and *Elimination* [6], [8]. In the Branch-and-Bound method, the set is subsequently partitioned in more and more refined parts (branching) over which bounds of an objective function value, or in our case, bounds on the constraint functions can be determined (bounding). Parts with lower bounds exceeding zero are deleted (pruning), since these parts of the domain can not contain solutions. Specifically for the problem of finding feasible solutions of (1), a possible algorithm based on bisection will be outlined. The method starts with a set $C_1 = S$. At every iteration the Branch-and-Bound method has a list Λ of subsets (partition sets) C_k of C_1. The method starts with C_1 as the first element and stops when the list is empty. For every set C_k in Λ, lower bounds g^L_{ik} of the constraint function values on C_k are determined. Generated subsets are not stored on Λ, if for one of the constraint functions $g^L_{ik} > 0$. In Section 3, a way of generating lower bounds is shown and a theoretical result is given that may improve the deletion by infeasibility. Moreover, to force theoretical convergence and due to some practical reasons one would like to discard partition sets that become too small. The branching concerns the further refinement of the partition. This means that one of the subsets is selected to be split into new subsets. A selection rule determines the subset to be split next.

Figure 2 sketches the idea of the bisection algorithm. It can be observed that

Algorithm 1 : Branch-and-Bound algorithm.

```
Inputs:                              – $g_i$: constraint functions
                                     – $\epsilon$: accuracy
Outputs: solution set $\Omega$ or "no solution"
Funct B & B Algorithm

 1. $\Lambda := \emptyset; r := n$                        $r$ = eval. vertices.
 2. $C_1 = S$, unit simplex
 3. for the vertices $v \in C_1$ Evaluate $g_i(v)$  $i = 1, \dots, m$
 4. evaluateSimplex(in: $C_1$, global: $\Lambda$)
 5. if $\Lambda = \emptyset$ STOP ; return "no solution"
 6. 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 $v_{new}$.
 7. Evaluate $g_i(v_{new})$  $i = 1, \dots, m$
 8. if $g_i(v_{new}) \leq 0$;  $i = 1, \dots, m$, store $v_{new}$ in $\Omega$
 9. EvaluateSimplex(in: $Cnew_1$, global: $\Lambda$) and EvaluateSimplex(in:$Cnew_2$,
    global: $\Lambda$)
10. $r := r + 1$, go to step 5
```

Algorithm 2 :Evaluate subset and decide to put on list based on lower bound.

```
Funct EvaluateSimplex ($C_k$); global $\Lambda$

1. if $size(C_k) > \epsilon$
2.   for $i = 1, \dots, n$ determine lower bounds $g_i^L(C_k)$
3.   if $g_i^L(C_k) \leq 0\ \forall i$
4.     store $C_k$ in $\Lambda$
```

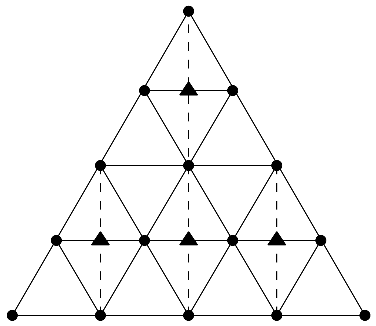

Figure 2: BB1: Bisection process 2-dim projection.

not only points on the regular grid are generated, but that the bisection also generates edges (the dotted lines) in at least one additional direction than the edges of the original simplex. A side effect was noticed where an implementation was applied in an industrial environment: the values of all generated feasible designs are a multiple of $(1/2)^K$, where K is an integer representing the depth of the search tree. An advantage of the bisection splitting is due to the shape of the partition sets. The length of the longest edge is at most twice the size of the shortest edge. Therefore the sets never get a needle shape. The *Size* of a simplex can be the euclidean length of the largest edge, but also other norms can be applied.

The number of simplices that is generated (and stored) in the worst case, depends on many aspects. We could derive an upper bound and a lower bound on the worst case performance. In the worst case, rules lead to splitting and storing simplices that have a size slightly larger than ϵ. After going $n(n-1)/2$ deeper in the search tree, at least all edges have been halved and the size of the simplex is less than half its original size. Suppose $Size(C_1) = 1$ (infinity norm distance). The maximum number K of halving the simplices is given by $1/2^K \leq \epsilon$, such that $K = \lceil(-ln\epsilon/ln(2))\rceil$, where $\lceil x \rceil$ is the lowest integer greater than or equal to x. 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 don't pass the size test. An overestimate for the worst case number of simplices on the list is:

$$2^{K \times n(n-1)/2-1} \tag{6}$$

where $K = \lceil(-ln\epsilon/ln(2))\rceil$. This analysis provides a guarantee that the algorithm is finite given an accuracy. For $\epsilon = 1\%$, it gives a bound of the order of 10^6 for $n = 3$ and 10^{44} for $n = 7$. This does not sound very encouraging nor realistic.

Let us now consider a lower bound on the worst case behaviour. Consider again the regular grid in Figure 1. Suppose an algorithm would generate with an accuracy of $\alpha = \frac{1}{M-1} < \epsilon$ all corresponding simplices of the regular mesh. We know that bisection has to generate more in the worst case. How many simplices would the algorithm generate? The number of simplices in the regular grid is

$$(M-1)^{n-1} \tag{7}$$

Formula (7) can be derived from volume considerations. The unit simplex represents a volume in $n-1$ dimensional space proportional to $(\sqrt{2})^{n-1}$. As the largest edge of a simplex within the regular grid has a size of $\frac{1}{M-1}$ times the unit simplex size, its volume is $\left(\frac{1}{M-1}\right)^{n-1}$ times smaller than that of the unit simplex. Also the number (7) turns out big; for $\epsilon = 1\%$, it gives 10^4 for $n = 3$ and 10^{12} for $n = 7$. We will observe in the experiments that practically the number is much lower. The real success of Branch-and-Bound depends on how early branches of the tree can be pruned.

3 Infeasibility check

A useful theorem is elaborated for the problem of finding feasible points of (1). Ways to find an underestimation g_i^L of g_i for a multidimensional set has been investigated by several researchers. Mladineo [9] showed that equating over x the cones

$$g(x) \geq g(v_j) - L\|x - v_j\| \tag{8}$$

for the $n+1$ vertices v_j of a simplicial region C in R^n, leads to a set of equalities that is not easy to solve. In [7] and [10] approximations of (8) are introduced over rectangular regions. Actually, a very precise lower bound is not needed, because we just have 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. In [4] the following lower bound was used on a simplicial set.

$$g^L = \max_j g(v_j) - L \times Size(C) \tag{9}$$

where $Size(C)$ is the maximum edge length of C. A few remarks can be made. First of all, notice again that it is not necessary to determine (9) if one of the vertices appears to be feasible with respect to function g. Moreover, [2] showed that the value of L can be refined, as one is moving in the lower dimensional simplex. This means that if L is determined by $\max_{x \in S} \|\nabla g(x)\|$, one can reduce the size of the gradient by projecting it in the unit simplex plane.

We now reconsider the lower bounding from the point of view of the feasibility test. If $g(v_j) > 0$, combining (8) and $g(v_j) - L\|x - v_j\| > 0$ gives that the spheres

$$\{x \in R^n, \|x - v_j\| \leq \frac{g(v_j)}{L}\} \tag{10}$$

cannot contain a feasible point with respect to function g. Consider again the set of inequalities (1) $i = 1, \ldots, m$, when all vertices are infeasible, i.e. at least for one of the functions, $g_i(v_j) > 0$. Around each vertex a sphere B_j is defined that cannot contain a feasible point

$$B_j = \{x \in R^n, \|x - v_j\|^2 < \varrho_j^2\},\ \varrho_j = \max_i \frac{g_i(v_j)}{L_i}, j = 1, \ldots, n \tag{11}$$

The test of $g^L > 0$ as defined in (9) becomes

$$\begin{array}{l} \max_i\{g_i^L = \max_j g_i(v_j) - L_i Size(C)\} > 0 \Rightarrow \\ \max_j \max_i\{g_i(v_j) - L_i Size(C)\} > 0 \Rightarrow \\ \max_j\{\varrho_j = \max_i \frac{g_i(v_j)}{L_i}\} > Size(C) \end{array} \tag{12}$$

which means that the largest infeasibility sphere should cover simplicial set C.

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.

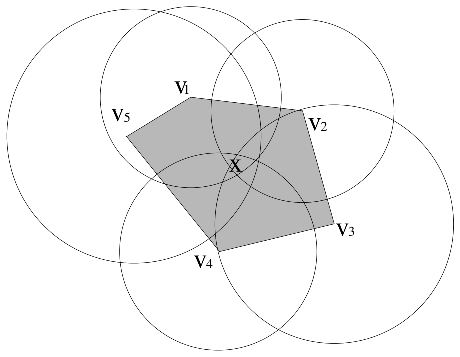

Figure 3: Polytope covered by raspberry set.

Theorem 1 *Given unit simplex S, a regular grid defined by grid points v_j with M points per axis and the problem of finding feasible solutions of (1) and (2). If*

$$\max_i \frac{g_i(v_j)}{L_i} > \frac{\sqrt{2}}{M-1} \quad \forall j \tag{13}$$

then S does not contain any feasible solution of (1) and (2).

Proof: Given the grid over S,

$$\forall x \in S \;\; \exists j \;\; \|x - v_j\| \leq \frac{\sqrt{2}}{M-1} \tag{14}$$

Combining (13) and (14) gives

$$\forall x \in S \;\; \exists i, j \;\; g_i(x) \geq g_i(v_j) - L_i\|x - v_j\| \geq g_i(v_j) - L_i\frac{\sqrt{2}}{M-1} > 0 \tag{15}$$

□

Thus far we made the translation from considering Lipschitz cones (8) to infeasibility spheres that handle well the concept of considering several constraint functions at the same time. For the Branch-and-Bound algorithm, the question has been shifted from lower bounding into covering by what we call the raspberry set $R(C) = \cup B_j$ (see [3]). In the algorithm, the bounding part can be substituted by a (in)feasibility check: does $R(C)$ cover set C, i.e. is $C \subset R(C)$? It is not straightforward to answer that test. To check this, one could look for a point in $C \setminus R(C)$. The following theorem, illustrated by Figure 3, shows that it is useful to look for a point that is covered by all infeasibility spheres.

Theorem 2 *Given vertices $v_1, \ldots, v_q$ of set $S =$conv$\{v_1, \ldots, v_q\}$, i.e. the vertices are extreme points of S. Let each vertex v_j be infeasible, i.e. $\max_i g_i(v_j) > 0$ with corresponding infeasibility sphere B_j defined by (11). If $\exists x \in \cap B_j \cap S$ then $S \subset \cup B_j$ and consequently no feasible point exists in S.*

Proof: The line of the proof follows that of a contradiction. Assume that $\exists y \in S$ such that $y \notin \cup B_j$. 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 $x \in \cap B_j$ and $y \notin \cup B_j$ follows that $\forall j \;\; \|y - v_j\|^2 \geq \varrho_j^2$ and $\|x - v_j\|^2 < \varrho_j^2$. Walking from one point to the other gives a mean-value result:

$$\begin{array}{l} \|y - v_j\|^2 = (x + r - v_j)^T(x + r - v_j) = \\ (x - v_j)^T(x - v_j) + 2r^T(x - v_j) + r^T r = \\ \|x - v_j\|^2 + 2r^T(x + \frac{1}{2}r - v_j) < \varrho_j^2 + 2r^T(z - v_j) \end{array} \tag{16}$$

Combining $\|y - v_j\|^2 \geq \varrho_j^2$ with (16) leads to the conclusion that

$$2r^T(z - v_j) > 0, \quad j = 1, \ldots, q \tag{17}$$

Equation (17) tells us that the assumption of the existence of y leads to the appearance of a point $z \in S$ and a direction r such that the directional derivative with respect to the squared distance function is positive with respect to all vertices $v_1, \ldots, v_q$. 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 S.

More exactly, function $f(a) = r^T a$ is linear in a and therefore attains its maximum over polytope S in one of the vertices v_p. For that vertex, $\max_{a \in S} 2r^T(a - v_p) = 0$ such that Equation (17) 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 S.

□

Theorem 2 shows that it is sufficient to find a point $x \in S$ that for each vertex v_j is closer than ϱ_j. One can write an algorithm for determining such a point. One can make a good guess by taking a weighted average

$$\xi = \frac{1}{\sum_j \frac{1}{\varrho_j}} \sum_j \frac{1}{\varrho_j} v_j \tag{18}$$

and test whether $\|\xi - v_j\|^2 < \varrho_j^2$ for each vertex v_j. Alternatively, in the experiments we used a point just within the interior of the smallest sphere. Let $p = \text{argmin}_j \varrho_j$. If ξ is not covered by the smallest sphere, $\|\xi - v_p\|^2 \geq \varrho_p^2$, we generate a trial point θ just in the smallest sphere defined by

$$\theta = v_p + (\frac{\varrho_p}{\|\xi - v_p\|} - \delta)(\xi - v_p) \tag{19}$$

where δ is a tiny accuracy number. In Section 4, experiments will be done with the two ways of checking the feasibility; either based on (12) (one Single sphere Cover (SC) test) or based on generating and checking the guess points ξ or θ (N spheres Cover (NC) test).

4 Cases

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 [2]). For the illustration, two three dimensional cases are taken from [4]. The advantage of three-dimensional cases is that we obtain graphical information on the feasible set, the regular grid and the rejected subsets. The first case, called RumCoke, is an illustrative example for outlining mixing with two quadratic constraints. The second case, Case-2, was taken from an industrial example having 5 quadratic constraints. The quadratic constraints are translated into Lipschitzian requirements via:

$$|g_i(x) - g_i(y)| = |(x-y)^T \nabla g_i(\frac{x+y}{2})| \leq L_i \|x-y| \tag{20}$$

where in [2] was shown that, because $x - y$ is only moving within the plane of the simplex, one can get a sharp bound on L_i by taking

$$L_i = \max_s \|\nabla g_i(x) - \frac{1}{n}\mathbf{1}^T \nabla g_i(x)\mathbf{1}\| \tag{21}$$

where $\mathbf{1}$ is the all ones vector. As the function in (21) is convex being the norm of a linear function in x, one simply obtains L_i by evaluating the gradient in the vertices of S, i.e. the unit vectors.

Figure 4 shows one of the iterations of the algorithm for Case-2. Each requirement is given a different colour. Each sphere (11) gets the colour of the corresponding requirement. The simplices currently on list Λ are given in grey. The areas that are deleted already, are given in dark blue when one of the spheres covered the complete simplex (SC test) and light blue when Theorem 2 was used generating either point ξ or moreover point θ (NC test). Focusing on the white simplex that is currently evaluated, one can observe that none of the spheres 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.

To illustrate the progress by applying Theorem 2 via the NC test, two versions of the algorithm are run. In the first version only the SC test is applied. The second version applies the NC test in case the SC test is not successful.

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 $K = 7$ times, such that the mesh size is $(\frac{1}{2})^7$, corresponding to $M = 129$ points per axis. This means that the corresponding grid contains 8385 points according to (7). To obtain the depth of $K = 7$, one should - working in Euclidean norm - apply in the algorithm an accuracy ϵ somewhere in between $\frac{\sqrt{2}}{128}$ and $\frac{\sqrt{2}}{64}$. We will use $\epsilon = \frac{\sqrt{2}}{100}$.

To measure the performance, we should keep in mind that a grid search would generate and evaluate 8385 points. The Branch-and-Bound algorithm does not only evaluate points, but also stores vertices (points) and simplices that need to be explored further. The algorithm finally generates 80 feasible

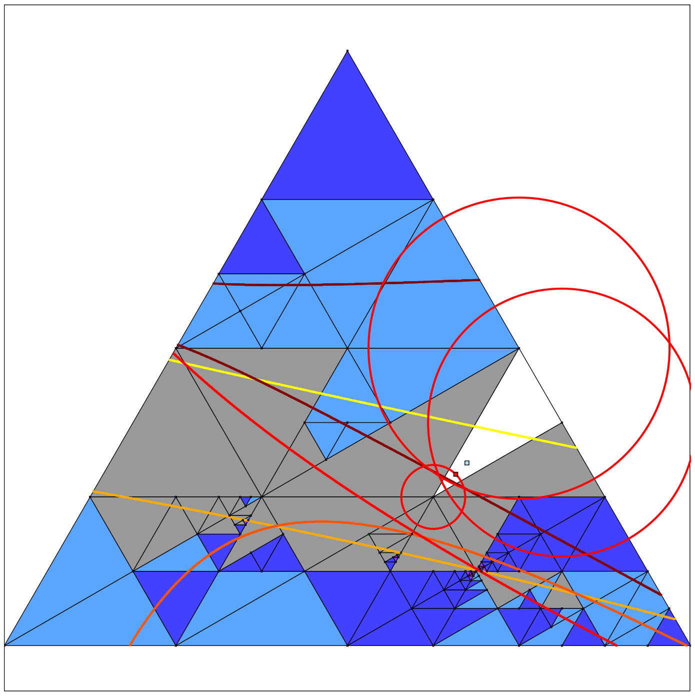

Figure 4: Iteration of the B&B algorithm for Case-2.

points for the RumCoke case and 281 for Case-2. An important performance indicator in Table 1 is the number of evaluated points to reach the result, that should be smaller than evaluating the complete grid. To reach this performance, the algorithm generates many simplices. For these smaller cases, the maximum storage needed does not grow out of hand. A generated and possibly stored simplex is either split or not explored further. It can be rejected, because it has been proven it does not contain feasible points with either SC or NC tests. Finally around the feasible area, the algorithm reaches the final accuracy at the bottom of the search tree and generates simplices of size smaller than ϵ.

The success of applying Theorem 2 can be read from savings on the number of points to be evaluated, the generated and tested simplices and from the required storage. The latter is of specific importance for practical cases in higher dimensions, as one has to store millions of subsets and corresponding vertices. The number of stored vertices only refer to those related to the stored simplices in Λ. The final feasible points are stored in Ω (see Algorithm 1) and do not require working memory. Moreover, consider the simplices that are not split further because their size is too small. The algorithm is much more successful in not to have to reach the bottom of the search tree applying the new infeasibility test. This means that bigger areas are thrown out. This can also be derived from the total number of generated simplices. The final result of the algorithm

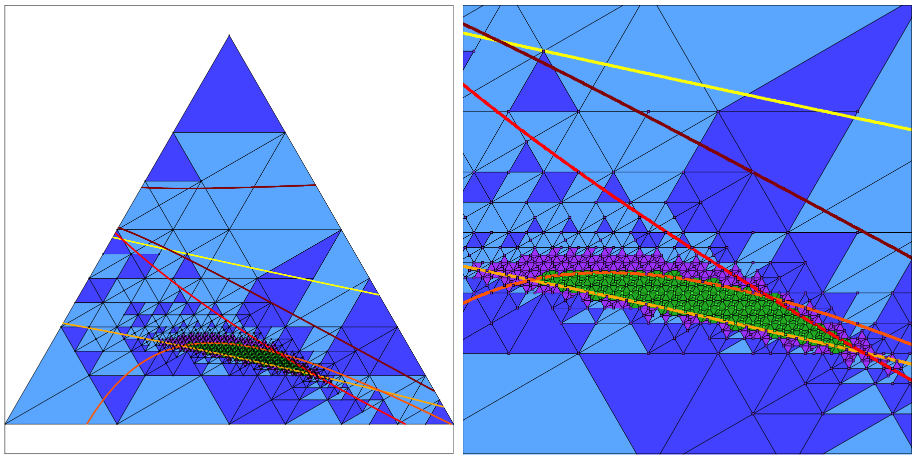

Figure 5: Final partitioning for Case-2 (left). Zoom on the feasible area for Case-2 and the generated feasible points (right).

Table 1: Results Branch-and-Bound algorithm for two different infeasibility tests, $\epsilon = \frac{\sqrt{2}}{100}$

	Gener. Simplex	Eval. Vertex	Maximum Storage Simplex	Maximum Storage Vertex	Infeas. Test SC	Infeas. Test NC	Size $< \epsilon$
RumCoke-SC	5246	2420	115	148	1792	–	832
RumCoke-NC	3036	1371	68	124	192	831	498
Case2-SC	3286	1304	52	329	941	–	703
Case2-NC	2432	910	33	312	296	306	615

is depicted in Figure 5.

The final simplices that are not split further, but that contain at least one of the 281 feasible vertices, are coloured green here. In such a way, one can see the feasible area. To zoom in on the cover of the feasible area by the generated points, one can have a look at the right picture in Figure 5. Here one can better observe all simplices that are not split further and do not contain feasible vertices, given by a purple colour.

5 Conclusions

A Branch-and-Bound algorithm has been described that generates feasible points on the unit simplex of a system of Lipschitz continuous inequalities. Among that points are feasible points on a regular grid over the simplex. It has been derived how many points have to be evaluated, if one would try all points on the regular grid.

A theorem has been derived that can help to check whether a polytope contains feasible points of a system of Lipschitz continuous inequalities. The theorem and suggestions for guess points have been used to improve the algorithm. The number of points evaluated by the algorithm and the storage requirements for generating the same result have been reduced considerably.

References

[1] Yu. Danilin and S. A. Piyavskii. An algorithm for finding the absolute minimum. *Theory of Optimal Decisions*, 2:25–37, 1967. (in Russian).

[2] Eligius M. T. Hendrix, L. G. Casado, and I. García. Branch-and-bound for the semi-continous quadratic mixture design problem SCQMDP. Technical Report 17, Mansholt Graduate School, http://www.sls.wau.nl/mi/mgs/publications, 2005. submitted to Journal of Global Optimization.

[3] Eligius M. T. Hendrix and Olivier Klepper. On uniform covering, adaptive random search and raspberries. *Journal of Global Optimization*, 18(2):143–163, 2000.

[4] Eligius M. T. Hendrix and Janos D. Pintér. An application of Lipschitzian global optimization to product design. *Journal of Global Optimization*, 1:389–401, 1991.

[5] R. Horst, P.M. Pardalos, and N.V. Thoai, editors. *Introduction to Global Optimization*, volume 3 of *Noncovex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, Holland, 1995.

[6] T. Ibaraki. Theoretical comparisons of search strategies in branch and bound algorithms. *Int. J. Comput. Inform. Sci.*, 5:315–344, 1976.

[7] C. C. Meewella and D. Q. Mayne. An algorithm for global optimization of Lipschitz continuous functions. *Journal of Optimization Theory and Applications*, 57:307–322, 1988.

[8] L. G. Mitten. Branch and bound methods: general formulation and properties. *Operations Research*, 18:24–34, 1970.

[9] R. H. Mladineo. An algorithm for finding the global maximum of a multimodal, multivariate function. *Mathematical Programming*, 34:188–200, 1986.

[10] Janos D. Pintér. Branch-and-bound algorithms for solving global optimization problems with Lipschitzian structure. *Optimization*, 19:101–110, 1988.

[11] B. O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal of Numerical Analysis*, 9:379–388, 1972.

[12] H. P. Williams. *Model Building in Mathematical Programming.* Wiley & Sons, Chichester, 1993.

Appendix

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

Algorithm 3 :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 $x_1, \ldots, x_n$
```

accuracy. To illustrate the non-exponential character the number of points on the regular grid is given for $M = 11$ and $M = 101$ in Table 2. The number of points results as well from the algorithm as from Equation (5)

Table 2: Number of regular grid points on the unit simplex

	n=2	n=3	n=4	n=5	n=6	n=7
M=11	11	66	286	1001	3003	8008
M=101	101	5151	176851	$4.6\ 10^6$	$9.7\ 10^7$	$1.7\ 10^9$