Improving the Efficiency of Genetic Algorithms for Linearly Constrained Optimization

Siegfried Vössner[*]
Department of Engineering- and Business Informatics
Graz University of Technology
Graz, Austria

Michael O'Sullivan[†, ‡]
Department of Engineering Science
University of Auckland
Auckland, New Zealand

Ulrich Kausch

Abstract

The efficiency of a Genetic Algorithm for constrained parameter optimization depends heavily on the ratio of feasible to infeasible area in its rectangular search space. We show an algorithm based on existing mathematical programming methods which improves this ratio assuming a set of linear constraints. We approximate the feasible area by a multidimensional ellipsoid and rotate the original search space parallel to its main axes. The minimum volume hyper rectangle we can wrap around the rotated constraint set gives us a new rectangular search space. In addition to that we propose to continue with a local search algorithm for fine tuning. To demonstrate the use of the proposed method, we perform test runs on randomly generated cases as well as on three selected examples.

[*] Research performed while visiting the Department of Engineering-Economic Systems and Operations Research (now Management Science and Engineering), Stanford University, during 1997.
[†] Corresponding author.
Email address: michael.osullivan@auckland.ac.nz (M. J. O'Sullivan)
[‡] Research performed while a PhD student in the Department of Engineering-Economic Systems and Operations Research (now Management Science and Engineering), Stanford University, during 1997.

Abstract

The efficiency of a Genetic Algorithm for constrained parameter optimization depends heavily on the ratio of feasible to infeasible area in its rectangular search space. We show an algorithm based on existing mathematical programming methods which improves this ratio assuming a set of linear constraints. We approximate the feasible area by a multidimensional ellipsoid and rotate the original search space parallel to its main axes. The minimum volume hyper rectangle we can wrap around the rotated constraint set gives us a new rectangular search space. In addition to that we propose to continue with a local search algorithm for fine tuning. To demonstrate the use of the proposed method, we perform test runs on randomly generated cases as well as on three selected examples.

Introduction

Genetic Algorithms (GAs) are used more and more for numerical optimization and conquered their place among other classical optimization techniques. Due to their robustness and flexibility they are mainly applied on complex, nonlinear, even stochastic optimization problems of moderate size. One important issue for an optimization algorithm is to handle constraints. The search space for a Genetic Algorithm has the form of an *n*-dimensional rectangle. However real world problems have more constraints than upper and lower bounds on the variables. In special cases it is possible to get rid of constraints using a problem specific coding scheme and problem specific operators to preserve feasibility of solutions (e.g.:[1]). In the other cases the feasible area is smaller than the search space, which makes the GA inefficient. This detail is especially important for problems where running time is a major concern. General applicable methods for handling constraints are: penalty functions and repair or mapping algorithms [2], [1], [3] to name the most important ones.
In this paper we present a method to improve the efficiency of handling linear constraints in general which can be applied in combination with all known constraint handling techniques for GAs.
We show a polynomial time algorithm to minimize the infeasible area in the search space by rotating the coordinate system before starting the GA. The GA then works more efficiently in a transformed coordinate system, still using constraint handling techniques.
The feasible area is a polyhedral set defined by the linear constraints (figure 1). We approximate this area by a multidimensional ellipsoid (figure 3) using Semi Definite Programming (SDP). The smallest rectangle one can wrap around an ellipsoid has sides parallel to its principle axes. Therefore we rotate the original search space parallel to the main axes of the ellipsoid. The smallest *n*-dimensional rectangle in the rotated system is then obtained via a sequence of linear programs (LPs). Since the replacement of the original polyhedral set by an ellipsoid is just an approximation we refine the search by a local gradient method. Finally we check the improvement by comparing the volume of the new box to the original one.
Minimizing the infeasible area typically has many local optima, so the overall optimal rotation can only be found by global optimization, which would require a large computational effort - especially in higher dimensions and with large constraint sets. The algorithm presented here uses a heuristic to get find the global optimum.
In the following paragraphs we briefly sketch the underlying theory for our algorithm, describe the algorithm itself and eventually show the benefits of the algorithm by some experimental results.

The Problem

Genetic Algorithms are often applied to numerical optimization problems. Without loss of generality we assume continuous variables in $\Re$. This mathematical program can be written in the following form:

$$\text{optimize} \quad z = f\left(x_1, x_2, \ldots, x_n\right),$$

where the objective function *z* is an arbitrary function in $\Re$. The search space $S \subseteq \Re^n$ for Genetic Algorithms is defined as an *n*-dimensional rectangle in $\Re^n$, defined by lower and upper bounds of the variables x:

$$lb_i \le x_i \le ub_i, \qquad 1 \le i \le n$$

Set $F \subseteq S$ defines the feasible part of the search space - the infeasible part *I* is the remaining set $I = S \setminus F$. If no additional constraints are given then F equals S. Otherwise F is defined by the following set of *m* constraints:

$$\left.\begin{matrix} g_1\left(x_1, x_2, \ldots x_n\right) \\ g_2\left(x_1, x_2, \ldots x_n\right) \\ \vdots \\ g_m\left(x_1, x_2, \ldots x_n\right) \end{matrix}\right\} \begin{matrix} \le \\ = \\ \ge \end{matrix} \left\{\begin{matrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{matrix}\right.$$

where $g_i \ \left(1 \le i \le m\right)$ are, like *z*, arbitrary functions in $\Re$.

Constraint Handling

As stated above, standard Genetic Algorithms without specific "feasibility preserving" coding schemes and operators can only search in an *n*-dimensional rectangle, which requires constraint-handling methods. Generally they can be grouped into the following two categories:

- methods which repair infeasible solutions
  Infeasible solutions are projected at the boundary or inside the feasible set *F* e.g. in [1], [3] or [4].

and

- methods based on penalty functions
  These methods add "penalties" to the objective function value for violating constraints e.g. in [2].

We do not discuss these methods here. One comprehensive survey is given in [5]. Our approach, presented in this paper, can be used in combination with all methods mentioned above.

Efficiency of a GA

The efficiency of Genetic Algorithms depends obviously on the size of the search space [6]. The more the search space can be reduced, the more efficient the algorithm will be. Furthermore some constraint handling techniques such as methods based on "repairing infeasible solutions" disturb the creation and propagation of building blocks (compare [7] or [6]) by disrupting schemes with their "corrections". If we have a constrained optimization problem we have to accept the disadvantages of these techniques. So we try to reduce the infeasible area *I* as much as possible.

Basics

In this section the basic underlying theory of our approach to reduce the size of the infeasible region in the search space is presented. We state an optimization problem with linear constraints and summarize the two mathematical programs used in our approach, LPs and SDPs.

Linear Constraints

The feasible region F is defined by a set of m linear in/equalities (linear constraints)

$$\left.\begin{matrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{matrix}\right\} \begin{matrix} \le \\ = \\ \ge \end{matrix} \left\{\begin{matrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{matrix}\right.$$

or equivalently $Ax \; \{\le / = / \ge\} \; b$ which is a convex, polyhedral set, hereafter referred to as polyhedron (see figure 1).

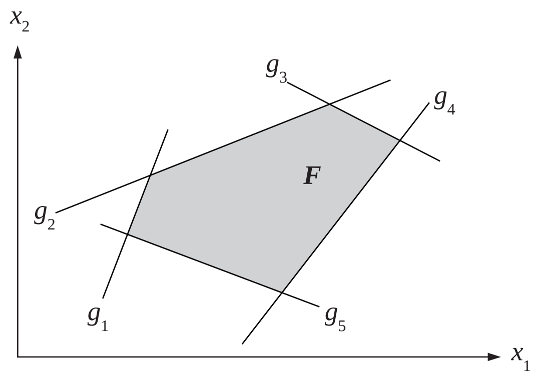

Figure 1, Convex Polyhedral Set.

However if there is a constraint of the form $a_i^{\mathrm{T}} x = b_i$ (where a_i is the i^{th} row of A) then the interior of the set is empty. The approach used in the sequel requires the polyhedron to have a non-empty interior. Therefore in the sequel it is assumed that the polyhedron consists of the set

$$\mathsf{P} = \{x \mid Ax \le b\}$$

where $x \in \Re^n$, $A \in \Re^{m \times n}$ and $b \in \Re^m$. The variable bounds are defined by the smallest hyper cube which contains P (see figure 2) and may be obtained via a sequence of linear programs.

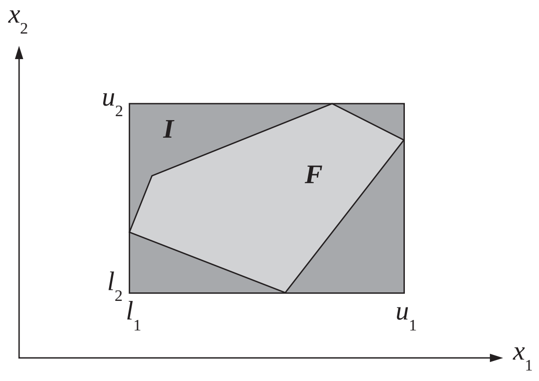

Figure 2, Smallest Enclosing Rectangle.

Linear Programming

Linear programming was first introduced by George Dantzig [8] and is widely used in many applications today. One form of a linear program is

$$\begin{array}{lll} \text{LP:} & \text{minimize} & c^{\mathrm{T}} x \\ & \text{subject to} & Ax \le b \end{array}$$

where $x \in \Re^n$, $c \in \Re^n$, $A \in \Re^{m\times n}$ and $b \in \Re^m$. In order to solve a linear program many different solution techniques can be used [8].

Semidefinite Programming - Maxdet Problem

Semidefinite programming (SDP) is a relatively new mathematical programming technique. For a detailed survey of the theory of SDP see e.g.: [9]. The general form of a SDP is

$$\begin{array}{lll} \text{SDP:} & \text{minimize} & c^{\mathrm{T}}x \\ & \text{subject to} & F(x) \geq 0 \end{array}$$

where $F(x) = F_0 + \sum_{i=1}^{n} x_i F_i$, $x \in \Re^n$, $c \in \Re^n$ and $F_i \in \Re^{m\times m}$. In this problem $F(x) \geq 0$ means that $F(x)$ is positive semi-definite (i.e. $z^{\mathrm{T}}F(x)z \geq 0$, $\forall\, z \in \Re^m$). SDPs can be solved using interior point methods similar to those developed for LPs. These methods are discussed in [10] and will not be repeated here.

One generalization of an SDP is the "determinant maximization problem". This problem, described by Vandenberghe and Boyd [11], is referred to as the "maxdet problem" and has the form

$$\begin{array}{lll} \text{maxdet:} & \text{minimize} & c^{\mathrm{T}}x + \log\det G(x)^{-1} \\ & \text{subject to} & G(x) > 0 \\ & & F(x) \geq 0 \end{array}$$

where x and c are defined as in the SDP, $G: \Re^n \to \Re^{l\times l}$ and $F: \Re^n \to \Re^{m\times m}$ are defined by

$$G(x) = G_0 + \sum_{i=1}^{n} x_i G_i\,,\ G_i \in \Re^{l\times l}$$

$$F(x) = F_0 + \sum_{i=1}^{n} x_i F_i\,,\ F_i \in \Re^{m\times m}$$

and $G(x) > 0$ means that $G(x)$ is positive definite. When $G(x) = 1$ the maxdet problem reverts to an SDP. Similar interior point methods to those used for SDPs can be applied to maxdet problems [11].

Maximum volume ellipsoid in a polyhedron

One example of the maxdet problem presented in [11] is the problem of maximizing the volume of an ellipsoid which lies in the interior of a polyhedron (see Figure 3).

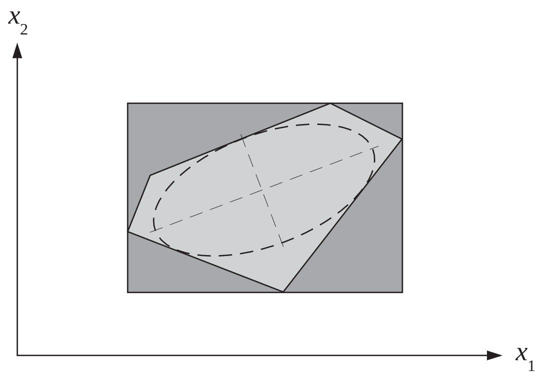

Figure 3, Maximum Volume Ellipsoid.

We use this method in our algorithm, so we briefly review it here.
As stated above, any polyhedron may be expressed by a set of linear inequalities $\mathsf{P} = \left\{x \,\middle|\, Ax \leq b\right\}$, where $x \in \Re^n$, $A \in \Re^{m\times n}$ and $b \in \Re^m$. Any ellipsoid may be represented by a matrix, C, and a vector, d, in the form

$$\mathsf{E} = \left\{Cy + d \,\middle|\, \|y\| \leq 1\right\}$$

where $C \in \Re^{n\times n}$ such that $C = C^{\mathrm{T}} > 0$ (i.e. C is positive definite and symmetric) and $d \in \Re^n$. The volume of the ellipsoid E is proportional to $\det C$. The problem of finding the maximum volume ellipsoid can be written as follows ("maxdet problem"):

$$\begin{array}{ll} \text{maximize} & \log\det C \\ \text{subject to} & C = C^{\mathrm{T}} > 0 \\ & \mathsf{E} \subseteq \mathsf{P} \end{array}$$

However

$$\begin{array}{lll} \mathsf{E} \subseteq \mathsf{P} & \Leftrightarrow & \sup\limits_{x\in\mathsf{E}} a_i^{\mathrm{T}}x \leq b_i\,,\ i = 1,\ldots,m \\ & \Leftrightarrow & \sup\limits_{\|y\|\leq 1} a_i^{\mathrm{T}}Cy + a_i^{\mathrm{T}}d \leq b_i\,,\ i = 1,\ldots,m \\ & \Leftrightarrow & \|Ca_i\| + a_i^{\mathrm{T}}d \leq b_i\,,\ i = 1,\ldots,m \\ & \Leftrightarrow & \begin{bmatrix} (b_i - a_i^{\mathrm{T}}d)I & Ca_i \\ a_i^{\mathrm{T}}C & b_i - a_i^{\mathrm{T}}d \end{bmatrix} \geq 0\,,\ i = 1,\ldots,m \end{array}$$

we have now formulated the problem as a maxdet problem

$$\begin{array}{ll} \text{maximize} & \log\det C \\ \text{subject to} & C = C^{\mathrm{T}} > 0 \end{array}$$

$$\begin{bmatrix} (b_i - a_i^{\mathrm{T}} d)I & Ca_i \\ a_i^{\mathrm{T}} C & b_i - a_i^{\mathrm{T}} d \end{bmatrix} \geq 0\ ,\ i = 1, \ldots, m$$

and it may be solved using the methods described in [11]. The matrix *C* is a rotation/scaling matrix which scales and then rotates the unit hyper sphere to conform to the ellipsoid required. The vector *d* is a translation vector which moves the ellipsoid from the origin to the interior of the polyhedron.

Approach

The ideas reviewed in the previous section are composed in an algorithm which attempts to rotate the search space so that the ratio of the feasible area to the total search space is increased. Consider the following example :

$$\mathrm{P} = \left\{ x \in \Re^2 \;\middle|\; \begin{array}{l} x_1 + x_2 \geq 1 \\ x_1 - x_2 \leq 1 \\ x_1 + x_2 \leq 4 \\ x_1 - x_2 \geq -1 \end{array} \right\}$$

This polyhedron is a rectangle rotated anticlockwise by $\pi/4$ radians. Calculating the upper and lower bounds in the original coordinate system gives a fairly inefficient search space. The dark gray area in figure 4a is the infeasible part of the search space.

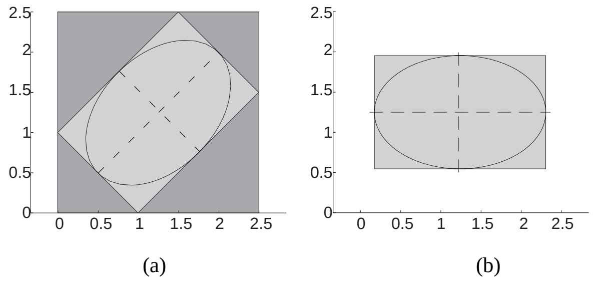

Figure 4, Example - Original (a) and rotated (b) coordinate system.

Obviously it would be a advantageous to rotate the coordinate system clockwise by $\pi/4$ radians. The search space found in the new coordinate system conforms exactly to the feasible region (light gray area in figure 4b).
In the following we show an algorithm to find the suitable angles for more common cases.

Finding the original search space

Finding the search space for linear constraints involves solving a sequence of LPs. Two LPs must be solved for each dimension of the problem, one to find the minimal value, the other to find the maximal value in that dimension. The linear constraints that constitute the polyhedron are

$$\mathrm{P} = \left\{ x \;\middle|\; Ax \leq b \right\} .$$

The two LPs for dimension *i* have the following formulations:

$$\begin{array}{lll} l_i : & \text{minimize} & x_i \\ & \text{subject to} & Ax \leq b \end{array}$$

and

$$\begin{array}{lll} u_i : & \text{minimize} & -x_i \\ & \text{subject to} & Ax \leq b \end{array}$$

They return the lower and upper bounds for the search space in dimension *i*. We call this part of the algorithm "shrinking". The problem now becomes one of finding the angles of rotation for the coordinate system which maximizes the ratio of feasible area to search space.

Find maximum volume ellipsoid

One heuristic for finding a good rotation for the coordinate system is to approximate the polyhedron by an ellipsoid. If the volume of the ellipsoid is maximal, but is still contained within the polyhedron then it approximates well its volume and shape. The accuracy of this approximation and therefore the quality of the heuristic depends on the shape of the polyhedral set.
Finding the maximum volume ellipsoid in a polyhedron belongs to the class of maxdet problems and can be solved by methods discussed in [11]. They yield an ellipsoid

$$\mathrm{E} = \left\{ Cy + d \;\middle|\; \|y\| \leq 1 \right\}$$

which consists of a rotation/scaling matrix *C* , and a translation vector *d* , which describe how to transform the unit hyper sphere into the required ellipsoid.

Finding the angles of the principal axes

The principal axes of the ellipsoid are given by the rotation of the original coordinate axes. The rotation/scaling matrix, *C*, both rotates and scales these axes. However by orthonormalizing *C* the scaling factor is removed and only the rotation *R* remains. The orthonormalized matrix, *R*, gives a basis transformation for the new coordinate system.

Rotating the coordinate system

To rotate the coordinate system the only change required is the basis change given by the orthonormalized matrix *R*. Rotation of the polyhedron

$$\mathrm{P} = \left\{ x \middle| Ax \le b \right\}$$

gives a new polyhedron

$$\mathrm{P}' = \left\{ x' \middle| ARx' \le b \right\}$$

which can then be used to find the new search space in the new coordinate system.

Finding the new search space

The new search space S' is found by shrinking an *n*-dimensional rectangle around the transformed polyhedron - a procedure similar to finding the original search space (see above). An *n*-dimensional rectangle represents the new search space in the new coordinate system.

Local Search

The rotation given by the principal axes of the maximum volume interior ellipsoid lies close to the possible minimum. The better the ellipsoid approximates the polyhedron, the closer the approach. The final approach is done by a local gradient search (e.g.: quasi-Newton [12]). The objective function for the search is the volume of the hyper rectangle calculated by solving the "shrinking" sub-problem. The variables represent the rotation angles (rotation) along the axes of the coordinate system.

Accept or Reject

Having defined a new search space the question has to be addressed, whether the new search space is more efficient than the original one. The comparison of the volume of the original search space V and the new search space V' yields an answer. As both of the search spaces are hyper rectangles these volumes are simple to calculate. As our approach is only a heuristic (it approximates the polyhedron by an ellipsoid) in some rare situations it can happen that the search space has deteriorated (i.e. increased in volume) after rotation. In such a case the original search space could not be reduced and is to be used without any modification.

Run Time Transformation Function for the GA

Eventually a run time transformation function needs to be provided for the GA. Rotating the coordinate system relative to the original system requires the re-transformation of the parameters x' generated in the rotated system when calling the fitness (objective) function. Since we have already computed the rotation matrix R and $x' = Rx$ we find

$$x = R^{-1} x'$$

by computing the inverse rotation matrix R^{-1}.

Estimate of Time Complexity

All our calculations are completed before the GA run starts (pre-processor), the time complexity of the algorithm shown here is independent from the time complexity of a GA. However finding an improved search space will improve the running time of the GA and so the efforts to obtain a new, smaller search space S' may be very beneficial.

Finding the new search space includes four steps :

- Finding the maximum ellipsoid inside the polyhedron via maxdet
- Rotation of the coordinate system
- Finding the new search space in the new coordinate system by "shrinking"
- Refining the search by using a local gradient search.

Vandenberghe and Boyd [11] showed that a general maxdet problem $x \in \Re^n$, $G: \Re^n \to \Re^{l \times l}$ and $F: \Re^n \to \Re^{m \times m}$ has a worst-case complexity of $\mathrm{O}\left(\sqrt{m}\right)$ Newton iterations. Each Newton iteration requires the computation of Newton directions which can be done in $\mathrm{O}\left(\left(m^2 + l^2\right)n^2\right)$ operations. Thus the total time complexity for solving the maxdet problem is $\mathrm{O}\left(\sqrt{m}\left(m^2 + l^2\right)n^2\right)$. However when finding the maximum ellipsoid inside the polyhedron ($m \times n$), the dimensions of the maxdet problem are $x \in \Re^{n(n+1)/2}$ (x is composed of the translation vector d and the symmetric part of the rotation matrix C), $G: \Re^{n(n+1)/2} \to \Re^{n \times n}$ and $F: \Re^{n(n+1)/2} \to \Re^{m(n+1) \times m(n+1)}$. Thus the running time in terms of the dimensions of the polyhedron is $\mathrm{O}\left(\sqrt{m(n+1)}\left(m^2 (n+1)^2 + n^2\right)\left(n(n+1)/2 + n\right)^2\right)$ which is $\approx \mathrm{O}\left(m^{5/2} n^{13/2}\right) \le \mathrm{O}\left(m^3 n^7\right)$. This means that the maximum volume interior ellipsoid is done in polynomial time in terms of the polyhedron dimensions. The rotation of the coordinate system is simply a case of a basis transformation which involves the multiplication of $A \in \Re^{m \times n}$ and $R \in \Re^{n \times n}$ which is $\mathrm{O}\left(mn^2\right)$. Shrinking in the new coordinate system requires to solve $2n$ LPs, each with a constraint matrix $A \in \Re^{m \times n}$. The time complexity for solving a single LP with the simplex method is $\mathrm{O}\left(2^m\right)$, so the time complexity of shrinking the box is $\mathrm{O}\left(n2^m\right)$. The function evaluation in the quasi-Newton search requires the hyper cube to be found by shrinking the box, so each function evaluation takes $\mathrm{O}\left(n2^m\right)$ time. The convergence rate of the gradient search (quasi-Newton) depends on the specific polyhedron.

The overall time complexity of the algorithm is dominated by the time needed to shrink the rectangle around the polyhedron. This has a time complexity of solving $2n$ LPs. The additional gradient search adds some calls to the "shrinking" problem. However if a LP solution method which runs in polynomial time is used then the algorithm may be run in polynomial time in terms of the dimensions of the polyhedron.

Just to give an idea on how fast the algorithm actually runs: Our test examples took less than 2 seconds to compute on a DEC 3000 workstation which is about 6 times slower than a personal computer with 166 MHz Pentium CPU, an example with 16 dimensions and 32 constraints took about 50 seconds on the same machine.

Results

The power of our method to minimize the search space, i.e. to increase the ratio of feasible to infeasible area was tested in some experiments. They were carried out in two dimensions, without loss of generality but with the possibility of plotting the search space. Three manually constructed cases are discussed and followed by randomly generated ones. The statistic we are considering is the ratio of the polygon area to the enclosing rectangle at three stages in our algorithm:

- before the rotation of the polygon
- after we rotated the polygon but before the local search
- after the local search.

The experiments were performed with a program written in Matlab [13] with calls to routines from [12] and [14].

Manually Constructed Difficult Cases

To demonstrate the behavior of our algorithm in extreme cases, three scenarios were constructed by hand, which are refered to as *Needle*, *Diamond*, and *Box*, and are drawn in figures 5 to 7 :

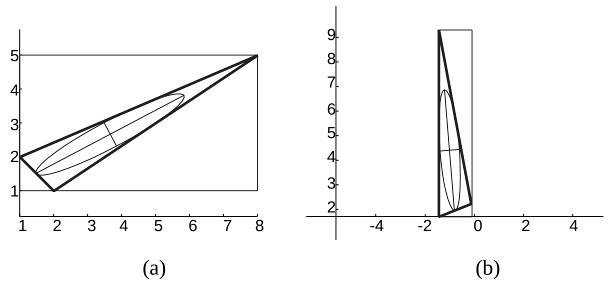

Figure 5, *Needle* before (a) and after rotation (b).

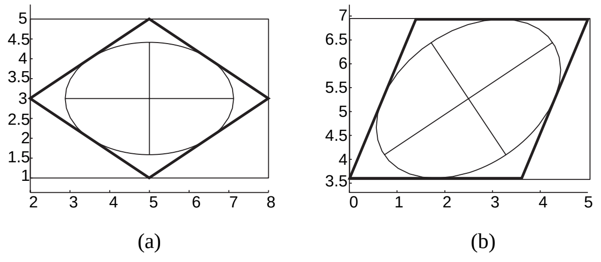

Figure 6, *Diamond* before (a) and after rotation (b).

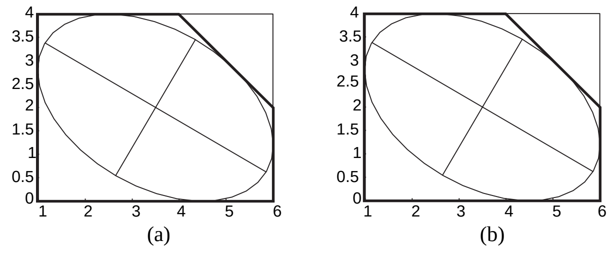

Figure 7, *Box* before (a) and after rotation (b).

The geometry of the *Needle* (figure 5a) favors our algorithm because obviously a rotation yields a much smaller infeasible area. The ratio of feasible to infeasible area has been remarkably increased from 0.1786 to 0.5015. For the diamond a similar result holds here (from 0.5000 to 0.7243). In addition the local search algorithm demonstrates the importance of the final rotation adjustment. Without this step, the polygon would not have been rotated at all because the principal axes of the ellipsoid are already parallel to the coordinate system (figure 6a). In the third example (*Box*, figure 7a) the algorithm behaves opposite to the way it did at the example of the *Diamond*: the initial rotation based on the ellipsoid did take place and the local search afterwards undid that turn because the ratio of feasible to infeasible area cannot be improved upon the initial state - since it was already optimal. The plots in figure 5b to 7b show the results of applying the algorithm to the three cases.

Tests On Randomly Generated Cases

Convex hulls containing 5, 7, 9, 11, and 13 points were considered to obtain some statistical significance for our results. In each category 40 samples were generated. The basic procedure to obtain the equations for the polyhedron is as follows: after randomly generating the points, an algorithm known as a *Graham Scan* [15] is used to find the convex hull. From the convex hull, we can compute the equations for the linear constraints easily.

The test statistic is the ratio of the polygon area to the enclosing rectangular area. Figure 8 depicts the distribution of the overall improvement of this ratio.

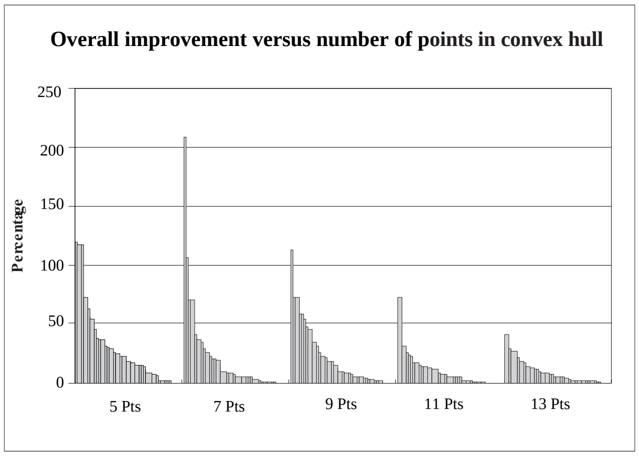

Figure 8, Distribution of the Overall Improvement.

It turns out that the improvement of the ratio is higher with fewer points. This is not surprising because the more points are contained in the convex hull, the more it tends to be rectangular and hence the percentage improvement by means of rotation decreases. In fact, in a few cases the rotation method does not yield any improvement.

The next issue we address is to show the necessity of a local search on top of the "maxdet" rotation (based on the orientation of the maximum volume ellipsoid). The following two graphs, which include only cases of more than 1% improvement, illustrate this. The improvement after the first ("maxdet") rotation is shown in figure 9.

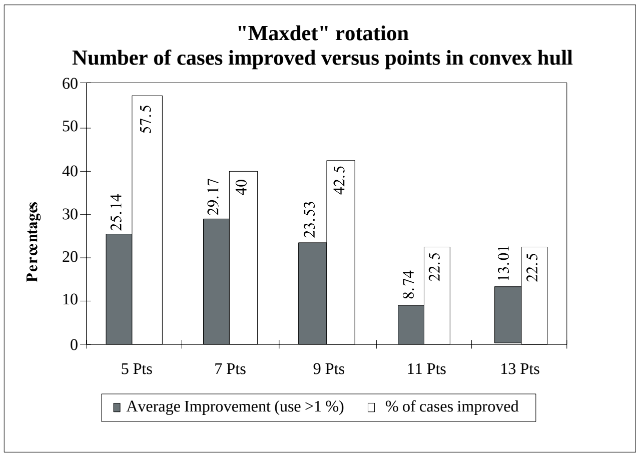

Figure 9, Initial Rotation Improvement.

Between 40% to 60% of all cases are improved by at least 23% for a convex hull consisting of 5, 7, or 9 points,. To yield an even higher improvement for more cases, a secondary rotation was added. This is essentially a steepest descent algorithm, which tries to improve upon the status quo. The results are shown in figure 10.

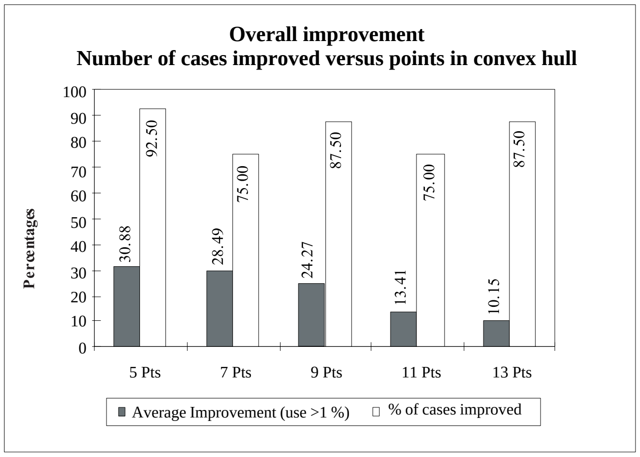

Figure 10, Overall Improvement.

The overall benefit of the second rotation (local search) is considerable: not only has the individual improvement increased in most cases, but also an improvement can be achieved for a much broader range of cases.

Discussion

The experimental results shown above, demonstrate that our algorithm yields a considerable reduction of the search space when doing parameter optimization with linear constraints. In addition the ratio of feasible to infeasible area is increased. Both facts can improve dramatically the overall behavior of a GA [6].

In addition to that two other strengths of this method are speed and scalability. The algorithm runs in polynomial time in the dimensions of the polyhedron and is also not limited by the dimensionality of the problem. Another advantage is that our algorithm does a pre-processing of the data and is therefore (with the exception of the run time transformation) independent of the optimization part itself. Any other constraint handling technique can still be applied to the modified data. The improvement that can be achieved by the proposed algorithm depends on how much the shape of the feasible set differs from a hyper rectangle: the greater the difference, the greater is the benefit.

The algorithm presented in this paper can be used for a all optimization problems with parameter optimization an linear constraints. It is fast, scalable and reduces the search of a GA considerably.

Future Work

In our future work we aim at handling nonlinear constraints as well and to adapt a GA for the local search part.

Acknowledgments

We would like to thank Stephen Boyd and Lieven Vandenberghe for their help, valuable discussions and permission to use their Matlab source code to solve the maxdet-program.

References

[1] Z. Michalewicz and G. A. Vignaux, “A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem,” *ORSA Journal on Computing*, vol. 3, pp. 307-316, 1991.

[2] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, “Some Guidelines for Genetic Algorithms with Penalty Functions,” presented at 3rd International Conference on Genetic Algorithms, 1989.

[3] W. Banzhaf, “Genotype-Phenotype-Mapping and Neutral Variation - A case study in Genetic Programming,” in *Parallel Problem Solving from Nature III*, Y. Davidor and e. al., Eds. Berlin: Springer, 1994.

[4] S. Vössner and G. Infanger, “A Hybrid Repair Technique for Constrained Parameter Optimization with Genetic Algorithms based on a NLP,” *unpublished manuscript*, 1996.

[5] Z. Michalewicz, “A Survey of Constraint Handling Techniques in Evolutionary Computation Methods,” presented at 4th Annual Conference on Evolutionary Programming, 1995.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.

[7] J. Holland, *Adaptation in Natural and Artificial Systems*, 2nd ed. Cambridge, Massachusetts: MIT Press, 1992.

[8] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press, 1963.

[9] F. Alizadeh, “Interior point methods in semidefinite programming with applications to combinatorial optimization,” *SIAM Journal on Optimization*, vol. 5, pp. 13-51, 1995.

[10] L. Vandenberghe and S. Boyd, “Semidefinite Programming,” *SIAM Review*, vol. 38, pp. 49-95, 1996.

[11] L. Vandenberghe, S. Boyd, and S.-P. Wu, “Determinant maximization with linear matrix inequality constraints,” *submitted to SIAM Journal on Matrix Analysis and Applications*, 1996.

[12] The MathWorks Inc., “Optimization TOOLBOX,” , 1994.

[13] The MathWorks Inc., “Matlab 4.2,” , 1994.

[14] L. Vandenberghe, S. Boyd, and S.-P. Wu, “maxdet,” 1996.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*: McGraw-Hill, 1990.