Towards Understanding Constraint-Handling Methods in Evolutionary Algorithms

Zbigniew Michalewicz
Dept. of Computer Science
Univ. of North Carolina
Charlotte, NC 28223, USA
zbyszek@uncc.edu, and
Polish Acad. of Sciences
Warsaw, Poland

Kalyanmoy Deb
Dept. of Mechanical Eng.
Indian Inst. of Technology
Kanpur, Pin 208 016
India
deb@iitk.ac.in

Martin Schmidt
Inst. of Computer Science
University of Aarhus
DK-8000 Aarhus C
Denmark
msc@bording.dk

Thomas J. Stidsen
Inst. of Computer Science
University of Aarhus
DK-8000 Aarhus C
Denmark
stidsen@daimi.au.dk

Abstract- The experimental results reported in many papers suggest that making an appropriate *a priori* choice of an evolutionary method for a nonlinear parameter optimization problem remains an open question. It seems that the most promising approach at this stage of research is experimental, involving a design of a *scalable* test suite of constrained optimization problems, in which many features could be easily tuned. Then it would be possible to evaluate merits and drawbacks of the available methods as well as test new methods efficiently.

In this paper we discuss a recently proposed test-case generator for constrained parameter optimization techniques. This generator is capable of creating various test cases with different characteristics and is very useful for analyzing and comparing different constraint-handling techniques.

1 Introduction

A vast majority of engineering optimization problems are constrained problems. The presence of constraints significantly affects the performance of any optimization algorithm, including evolutionary search methods [2]. This paper focuses on the issue of evaluating constraints handling methods, as the advantages and disadvantages of various methods are not well understood.

The general nonlinear programming (NLP) problem is to find $\vec{x}$ to

$$\text{optimize } f(\vec{x}), \quad \vec{x} = (x_1, \ldots, x_n) \in R^n, \qquad (1)$$

where $\vec{x} \in \mathcal{F} \subseteq \mathcal{S}$. The *objective function* f is defined on the *search space* $\mathcal{S} \subseteq R^n$ and the set $\mathcal{F} \subseteq \mathcal{S}$ defines the *feasible region*. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in R^n (domains of variables defined by their lower and upper bounds):

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n,$$

whereas the feasible region $\mathcal{F} \subseteq \mathcal{S}$ is defined by a set of p additional constraints ($p \geq 0$):

$$g_j(\vec{x}) \leq 0, \text{ for } j = 1, \ldots, q, \text{ and } h_j(\vec{x}) = 0, \text{ for } j = q+1, \ldots, p.$$

At any point $\vec{x} \in \mathcal{F}$, the constraints g_j that satisfy $g_j(\vec{x}) = 0$ are called the *active* constraints at $\vec{x}$.

The NLP problem is intractable: it is impossible to develop a deterministic method for the NLP in the global optimization category, which would be better than the exhaustive search [1]. This makes a room for evolutionary algorithms, extended by some constraint-handling methods. Indeed, during the last few years, several evolutionary algorithms (which aim at complex objective functions, e.g., non differentiable or discontinuous) have been proposed for the NLP; a recent survey paper [5] provides an overview of these algorithms.

During the last few years several methods were proposed for handling constraints by genetic algorithms for parameter optimization problems. These methods can be grouped into five categories: (1) methods based on preserving feasibility of solutions, (2) methods based on penalty functions, (3) methods which make a clear distinction between feasible and infeasible solutions, (4) methods based on decoders, and (5) other hybrid methods (for survey, see [5]). The general way of dealing with constraints — whatever the optimization method — is by penalizing infeasible points. However, there are no guidelines on designing penalty functions. Some suggestions for evolutionary algorithms are given in [7], but they do not generalize. Other techniques that can be used to handle constraints are more or less problem dependent. For instance, the knowledge about linear constraints can be incorporated into specific operators [4], or a repair operator can be designed that projects infeasible points onto feasible ones [6].

Moreover, the experimental results reported in many papers suggest that making an appropriate *a priori* choice of an evolutionary method for a nonlinear parameter optimization problem remains an open question. It seems that the most promising approach at this stage of research is experimental, involving a design of a *scalable* test suite of constrained optimization problems, in which many features could be easily tuned. Then it would be possible to evaluate merits and drawbacks of the available methods as well as test new methods efficiently. Section 2 discusses further the need for a such scalable test suite, whereas section 3 presents a particular test case generator proposed recently [3]. Section 4 provides an example of the use of this test case generator for evaluation of a particular constraint-handling method. Section 5 concludes the paper.

0-7803-5536-9/99/$10.00 ©1999 IEEE

2 A need for a test case generator

It is not clear what characteristics of a constrained problem make it difficult for an evolutionary technique (and, as a matter of fact, for any other optimization technique). Any problem can be characterized by various parameters; these may include the number of linear and nonlinear constraints, the number of equality constraints, the number of active constraints, the ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ between the sizes of feasible search space and the whole search space, the type of the objective function (the number of variables, the number of local optima, the existence of derivatives, etc). In [5] eleven test cases for constrained numerical optimization problems were proposed ($G1$–$G11$). These test cases include objective functions of various types (linear, quadratic, cubic, polynomial, nonlinear) with various numbers of variables and different types (linear inequalities, nonlinear equations and inequalities) and numbers of constraints. The ratio ρ between the size of the feasible search space $\mathcal{F}$ and the size of the search space $\mathcal{S}$ for these test cases vary from 0% to almost 100%; the topologies of feasible search spaces are also different.

The results of many tests did not provide meaningful conclusions, as no single parameter (number of linear, nonlinear, active constraints, the ratio ρ, type of the function, number of variables) proved to be significant as a major measure of difficulty of the problem. For example, many tested methods approached the optimum quite closely for the test cases $G1$ and $G7$ (with $\rho = 0.0111\%$ and $\rho = 0.0003\%$, respectively), whereas most of the methods experienced difficulties for the test case $G10$ (with $\rho = 0.0010\%$). Two quadratic functions (the test cases $G1$ and $G7$) with a similar number of constraints (9 and 8, respectively) and an identical number (6) of active constraints at the optimum, gave a different challenge to most of these methods. Also, several methods were quite sensitive to the presence of a feasible solution in the initial population. Possibly a more extensive testing of various methods was required.

Not surprisingly, the experimental results of [5] suggested that making an appropriate *a priori* choice of an evolutionary method for a nonlinear optimization problem remained an open question. It seems that more complex properties of the problem (e.g., the characteristic of the objective function together with the topology of the feasible region) may constitute quite significant measures of the difficulty of the problem. Also, some additional measures of the problem characteristics due to the constraints might be helpful. However, this kind of information is not generally available. In [5] the authors wrote:

> "It seems that the most promising approach at this stage of research is experimental, involving the design of a scalable test suite of constrained optimization problems, in which many [...] features could be easily tuned. Then it should be possible to test new methods with respect to the corpus of all available methods."

Clearly, there is a need for a parameterized test-case generator which can be used for analyzing various methods in a systematic way (rather than testing them on a few selected test cases; moreover, it is not clear whether addition of a few extra test cases is of any help).

In this paper we propose such a test-case generator for constrained parameter optimization techniques. This generator is capable of creating various test cases with different characteristics:

- problems with different value of ρ: the relative size of the feasible region in the search space;
- problems with different number and types of constraints;
- problems with convex or non-convex objective function, possibly with multiple optima;
- problems with highly non-convex constraints consisting of (possibly) disjoint regions.

All this can be achieved by setting a few parameters which influence different characteristics of the optimization problem. Such a test-case generator should be very useful for analyzing and comparing different constraint-handling techniques.

There were some attempts in the past to propose a test case generator for unconstrained parameter optimization [9, 10]. We are also aware of one attempt to do so for constrained cases; in [8] the author proposes so-called stepping-stones problem defined as:

$$\text{objective: maximize } \sum_{i=1}^{n}(x_i/\pi + 1),$$

where $-\pi \leq x_i \leq \pi$ for $i = 1, \ldots, n$ and the following constraints are satisfied:

$$e^{x_i/\pi} + cos(2x_i) \leq 1 \text{ for } i = 1, \ldots, n.$$

Note that the objective function is linear and that the feasible region is split into 2^n disjoint parts (called stepping-stones). As the number of dimensions n grows, the problem becomes more complex. However, as the stepping-stones problem has one parameter only, it can not be used to investigate some aspects of a constraint-handling method.

3 The test case generator

As explained in the previous section, it is of great importance to have a parameterized generator of test cases for constrained parameter optimization problems. By changing values of some parameters it would be possible to investigate merits/drawbacks (efficiency, cost, etc) of many constraint-handling methods. Many interesting questions could be addressed:

- how the efficiency of a constraint-handling method changes as a function of the number of disjoint components of the feasible part of the search space?

- how the efficiency of a constraint-handling method changes as a function of the ratio between the sizes of the feasible part and the whole search space?
- what is the relationship between the number of constraints (or the number of dimensions, for example) of a problem and the computational effort of a method?

and many others. In [3] a parameterized test-case generator $\mathcal{TCG}$ was proposed:

$$\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu);$$

which control:
n – the number of variables of the problem
w – the number of optima in the search space
λ – the number of constraints (inequalities)
α – the connectedness of the feasible search regions
β – the ratio of the feasible to total search space
μ – the ruggedness of the fitness landscape
The ranges and types of the parameters are:

$n \geq 1$; integer $\quad w \geq 1$; integer $\quad 0 \leq \lambda \leq 1$; float
$0 \leq \alpha \leq 1$; float $\quad 0 \leq \beta \leq 1$; float $\quad 0 \leq \mu \leq 1$; float

The general idea behind this test case generator was to divide the search space $\mathcal{S}$ into a number of disjoint subspaces $\mathcal{S}_k$ and to define a unimodal function f_k for every $\mathcal{S}_k$. Thus the objective function G is defined on the search space $\mathcal{S} = \prod_{i=1}^{n}[0, 1)$ as follows:

$$G(\vec{x}) = f_k(\vec{x}) \text{ iff } \vec{x} \in \mathcal{S}_k.$$

The total number of subspaces $\mathcal{S}_k$ is equal to w^n, as each dimension of the search space is divided into w equal length segments (for exact definition of a subspace $\mathcal{S}_k$, see [3]). This number indicates also the total number of local optima of function G. For each subspace $\mathcal{S}_k$ $(k = 0, \ldots, w^n - 1)$ a function f_k is defined:

$$f_k(x_1, \ldots, x_n) = a_k \left(\Pi_{i=1}^{n}(u_i^k - x_i)(x_i - l_i^k)\right)^{\frac{1}{n}} \quad (2)$$

where l_i^k and u_i^k are the boundaries of the k-th subspace for the i-th dimension. The constants a_k are defined as follows:

$$a_k = \begin{cases} 4w^2(1 - \alpha^2\beta^2)^{k'[(1-\mu)+\mu/\log_2(wn-n+1)]-1} \\ \qquad \text{if } \alpha\beta > 0 \\ 4w^2\left(\frac{(\mu-1)k''}{n(w-1)} + 1\right) \\ \qquad \text{if } \alpha\beta = 0, \end{cases} \quad (3)$$

where

$$k' = \log_2(\textstyle\sum_{i=1}^{n} q_{i,k} + 1), \text{ and } k'' = \textstyle\sum_{i=1}^{n} q_{i,k},$$

where $(q_{1,k}, \ldots, q_{n,k})$ is a n-dimensional representation of the number k in w-ary alphabet [1]. Additionally, to remove any predefined and fixed pattern from the generated test cases, an additional mechanism: a random permutation of f_k's was used.

The third parameter λ of the test-case generator is related to the number m of constraints of the problem, as the feasible part of the search space $\mathcal{S}$ is defined by means of m double inequalities $(1 \leq m \leq w^n)$:

$$r_1^2 \leq c_k(\vec{x}) \leq r_2^2, \quad k = 0, \ldots, m-1, \quad (4)$$

where $0 \leq r_1 \leq r_2$ and each $c_k(\vec{x})$ is a quadratic function:

$$c_k(\vec{x}) = (x_1 - p_1^k)^2 + \ldots + (x_n - p_n^k)^2,$$

where $p_i^k = (l_i^k + u_i^k)/2$. These m double inequalities define m feasible parts $\mathcal{F}_k$ of the search space:

$$\vec{x} \in \mathcal{F}_k \text{ iff } r_1^2 \leq c_k(\vec{x}) \leq r_2^2,$$

and the overall feasible search space $\mathcal{F} = \cup_{k=0}^{m-1}\mathcal{F}_k$. In the following, we refer to each of $\mathcal{F}_k$'s as a ring; note that such a ring is bounded by two spheres with radii r_1 and r_2. Note, the interpretation of constraints here is different than the one in the standard definition of the NLP problem. Here the feasible search space is defined as a *union* (not intersection) of all double constraints. In other words, a point $\vec{x}$ is feasible if and only if there exist an index $0 \leq k \leq m-1$ such that double inequality (4) is satisfied.

The parameter $0 \leq \lambda \leq 1$ determines the expected number m of rings as follows:

$$m = \lfloor \lambda(w^n - 1) + 1 \rfloor. \quad (5)$$

These rings are distributed randomly among subspaces $\mathcal{S}_k$. Clearly, $\lambda = 0$ and $\lambda = 1$ imply $m = 1$ and $m = w^n$, i.e., minimum and maximum number of rings, respectively.

The parameters α and β define the radii r_1 and r_2:

$$r_1 = \frac{\alpha\beta\sqrt{n}}{2w}, \quad r_2 = \frac{\alpha\sqrt{n}}{2w}. \quad (6)$$

These parameters determine the topology of the feasible part of the search space.

If $\alpha\beta > 0$, the function G has 2^n global maxima points, all on the inner sphere in the permuted subspace $\mathcal{S}_0$. For any global solution $(x_1, \ldots, x_n)$, $x_i = 1/(2w) \pm \alpha\beta/(2w)$ for all $i = 1, 2, \ldots, n$. The function values at these solutions are always equal to one. On the other hand, if $\alpha\beta = 0$, the function G has either one global maximum (if $\mu < 1$) or m maxima points (if $\mu = 1$), one in each of the permuted subspaces $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$. If $\mu < 1$, the global solution $(x_1, \ldots, x_n)$ is always at

$$(x_1, \ldots, x_n) = (1/(2w), 1/(2w), \ldots, 1/(2w))$$

which is the center of the subspace $\mathcal{S}_0$ before the random permutation is performed. Figure 1 displays the final landscape for test case $\mathcal{TCG}(2, 5, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$.
The interpretation of the six parameters of the test-case generator $\mathcal{TCG}$ is as follows:

[1] For $w > 1$. If $w = 1$ the whole search space consists of one subspace $\mathcal{S}_0$ only. In this case $a_0 = 4/(1 - \alpha^2\beta^2)$.

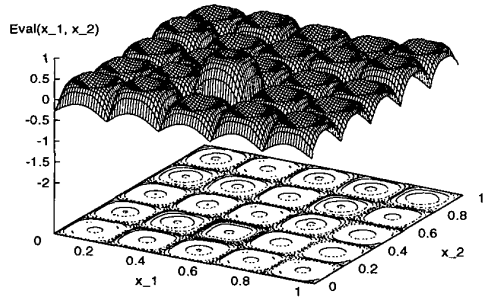

Figure 1: Final landscape for the test case $\mathcal{TCG}(2, 5, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$. Contours with function values ranging from -0.2 to 1.0 at a step of 0.2 are drawn at the base. The penalty coefficient $W = 1$ (see equation (7))

1. **Dimensionality** n: By increasing the parameter n the dimensionality of the search space can be increased.

2. **Multimodality** w: By increasing the parameter w the multimodality of the search space can be increased. For the unconstrained function, there are w^n locally maximum solutions, of which one is globally maximum. For the constrained test function with $\alpha\beta > 0$, there are $(2w)^n$ different locally maximum solutions, of which 2^n are globally maximum solutions.

3. **Constraintness** λ: By increasing the parameter λ the number m of constraints is increased.

4. **Connectedness** α: By reducing the parameter α (from 1 to $1/\sqrt{n}$ and smaller), the connectedness of the feasible subspaces can be reduced. When $\alpha < 1/\sqrt{n}$, the feasible subspaces $\mathcal{F}_k$ are completely disconnected. Additionally, parameter α (with fixed β) influences the proportion of the feasible search space to the complete search space (ratio ρ).

5. **Feasibility** β: By increasing the ratio β the proportion of the feasible search space to the complete search space can be reduced. For β values closer to one, the feasible search space becomes smaller and smaller. These test functions can be used to test an optimizer's ability to find and maintain feasible solutions.

6. **Ruggedness** μ: By increasing the parameter μ the function ruggedness can be increased (for $\alpha\beta > 0$). A sufficiently rugged function will test an optimizer's ability to search for the globally constrained maximum solution in the presence of other almost equally significant locally maxima.

Increasing each of the above parameters (except α) and decreasing α will cause an increased difficulty for any optimizer. However, it is difficult to conclude which of these factors most profoundly affects the performance of an optimizer. Thus, it is recommended that the user should first test his/her algorithm with the simplest possible combination of the above parameters (small n, small w, small μ, large α, small β, and small λ). Thereafter, the parameters may be changed in a systematic manner to create more difficult test functions. The most difficult test function is created when large values of parameters n, w, λ, β, and μ together with a small value of parameter α are used.

For details of the test case generator the reader is referred to [3].

4 An example

To test the usefulness of the $\mathcal{TCG}$, a simple steady-state evolutionary algorithm was developed. We have used a constant population size of 100 and each individual is a vector $\vec{x}$ of n floating-point components. Parent selection was performed by a standard binary tournament selection. An offspring replaces the worse individual of a binary tournament. One of three operators was used in every generation (the selection of an operator was done accordingly to constant probabilities 0.5, 0.15, and 0.35, respectively):

- Gaussian mutation: $\vec{x} \leftarrow \vec{x} + \vec{N}(0, \vec{\sigma})$, where $\vec{\sigma} = (1/(2\sqrt{n}), \ldots, 1/(2\sqrt{n}))$ for all experiments reported in this section.

- uniform crossover: $\vec{z} \leftarrow (z_1, \ldots, z_n)$, where each z_i is either x_i or y_i (with equal probabilities), where $\vec{x}$ and $\vec{y}$ are two selected parents.

- heuristic crossover: $\vec{z} = r \cdot (\vec{x} - \vec{y}) + \vec{x}$, where r is a uniform random number between 0 and 0.25, and the parent $\vec{x}$ is not worse than $\vec{y}$.

The termination condition was to quit the evolutionary loop if an improvement in the last $N = 10,000$ generations was smaller than a predefined $\epsilon = 0.001$.

Now the $\mathcal{TCG}$ can be used for investigating the merits of any constraint-handling methods. For example, one of the simplest and the most popular constraint-handling method is based on static penalties; let us define a particular static penalty method as follows:

$$eval(\vec{x}) = f(\vec{x}) + W \cdot v(\vec{x}), \tag{7}$$

where f is an objective function, W is a constant (penalty coefficient) and function v measures the constraint violation. (Note that only one double constraint is taken into account as the $\mathcal{TCG}$ defines the feasible part of the search space as a union of all m double constraints.)

The penalty coefficient W was set to 10 and the value of v for any $\vec{x}$ is defined by the following procedure:

find k such that $\vec{x} \in \mathcal{S}_k$
set $C = (-2w)/\sqrt{n}$
if the whole $\mathcal{S}_k$ is infeasible **then** $v(\vec{x}) = -1$
else
begin
calculate distance $Dist$ between $\vec{x}$ and
the center of the subspace $\mathcal{S}_k$
if $Dist < r_1$ **then** $v(\vec{x}) = C \cdot (r_1 - Dist)$
else if $Dist > r_2$
then $v(\vec{x}) = C \cdot (Dist - r_2)$
else $v(\vec{x}) = 0$
end

Thus the constraint violation measure v returns -1, if the evaluated point is in fully infeasible subspace (i.e., subspace without a ring); 0, if the evaluated point is feasible; or some number q from the range $[-1, 0)$, if the evaluated point is infeasible, but the corresponding subspace *has* a ring. It means that the point $\vec{x}$ is either inside the smaller sphere or outside the larger one. In both cases q is a scaled negative distance of this point to the boundary of the closest sphere. Note that the scaling factor C guarantees that $0 < q \leq 1$. Note, that the values of the objective function for feasible points of the search space stays in the range $[0, 1]$; the value at the global optimum is always 1. Thus, both the function and constraint violation values are normalized in the range [0,1].

As the test case generator has 6 parameters, it is difficult to discuss fully their interactions in a short paper. Thus we have selected a single point from the $\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$ parameter search space:

$$n = 2;\ w = 10;\ \lambda = 1.0;$$
$$\alpha = 0.9/\sqrt{n};\ \beta = 0.1;\ \mu = 0.1,$$

and varied one parameter at the time. In each case two figures summarize the results (which display averages of 10 runs):

1. all even numbered figures display the fitness value of the best individual at the end of the run (continuous line), the average height and the lowest height among all feasible optima (broken lines).

2. all odd numbered figures rescale the figure they follow: the fitness value of the best individual at the end of the run (continuous line) and the average height among feasible peaks in the landscape (broken line) are displayed as fractions between 0 (which corresponds to the height of the lowest peak) and 1 (which corresponds to the height of the highest peak).

Figures 2 and 3 display the absolute performance and relative performance respectively, of our static penalty method on the $\mathcal{TCG}$ while n is varied between 1 and 6. It is clear that an increase of n (number of dimensions) reduces the efficiency of the algorithm: the value of returned solution approaches the value of the average peak height in the landscape.

Figures 4 and 5 display the absolute performance and relative performance respectively, of a static penalty method on

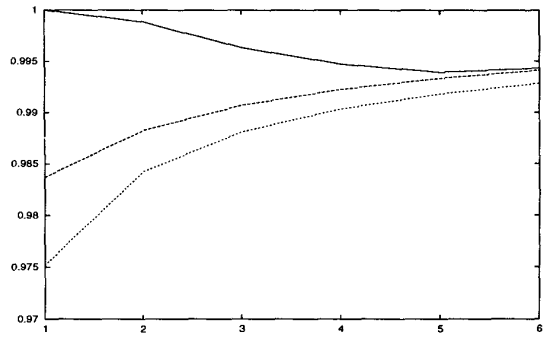

Figure 2: A run of the system for the $\mathcal{TCG}(n, 10, 1.0, 0.9/\sqrt{n}, 0.1, 0.1)$, absolute performance.

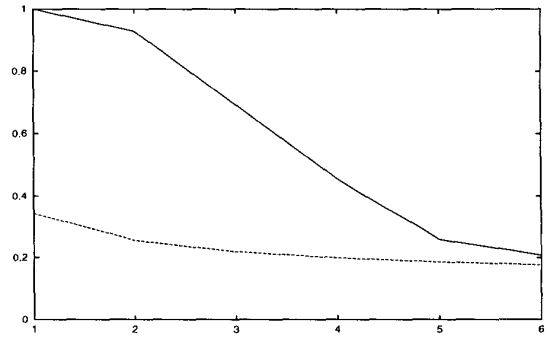

Figure 3: A run of the system for the $\mathcal{TCG}(n, 10, 1.0, 0.9/\sqrt{n}, 0.1, 0.1)$, relative performance.

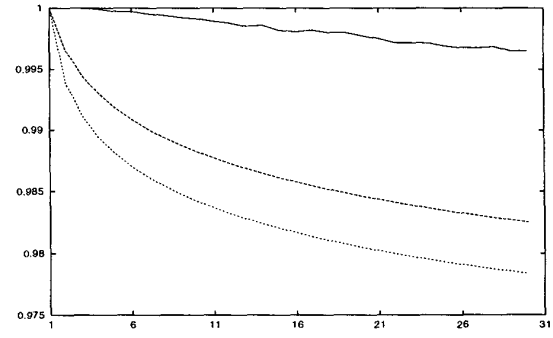

Figure 4: A run of the system for the $\mathcal{TCG}(2, w, 1.0, 0.9/\sqrt{2}, 0.1, 0.1)$, absolute performance.

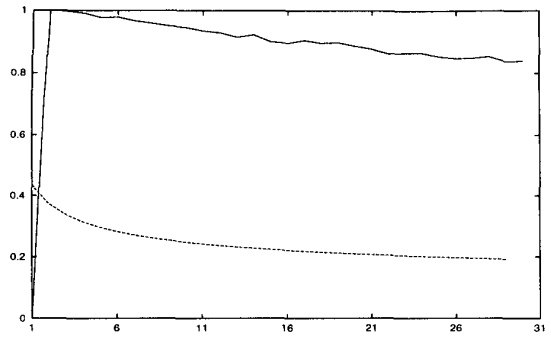

Figure 5: A run of the system for the $\mathcal{TCG}(2, w, 1.0, 0.9/\sqrt{2}, 0.1, 0.1)$, relative performance.

the $\mathcal{TCG}$ while w is varied between 1 and 30 (for $w = 30$ the objective function has $w^n = 900$ peaks). An increase of w decreases the performance of the algorithm, but not to extend we have seen in the previous case (increase of n). The reason is that while $w = 10$ (Figures 2 and 3), the number of peaks grows as 10^n (for $n = 3$ there are 1000 peaks), whereas for $n = 2$ (Figures 4 and 5), the number of peaks grows as w^2 (for $w = 30$ there are 900 peaks only).

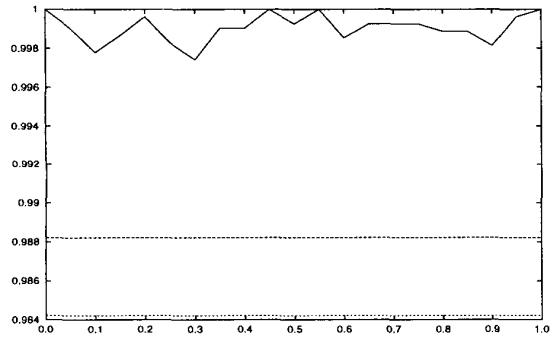

Figure 6: A run of the system for the $\mathcal{TCG}(2, 10, \lambda, 0.9/\sqrt{2}, 0.1, 0.1)$, absolute performmance.

Figures 6 and 7 display the absolute performance and relative performance respectively, of a static penalty method on the $\mathcal{TCG}$ while λ is varied between 0 and 1. It seems that the number of constraints does not influence the performance of the system.

Figures 8 and 9 display the absolute performance and relative performance, of a static penalty method on the $\mathcal{TCG}$ while α is varied between 0 and $1/\sqrt{2}$. Clearly, larger values of α improve the results of the algorithm, as it is easier to locate a feasible solution. Note an anomaly in the graph for $\alpha = 0$; this is due to the equation (3), which provides with a different formula for a_k's when $\alpha\beta = 0$.

Figures 10 and 11 display the absolute performance and the relative performance respectively, of a static penalty

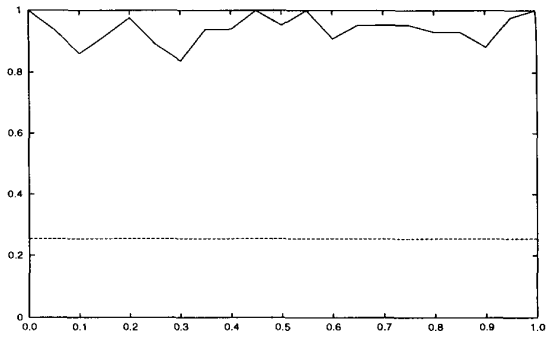

Figure 7: A run of the system for the $\mathcal{TCG}(2, 10, \lambda, 0.9/\sqrt{2}, 0.1, 0.1)$, relative performmance.

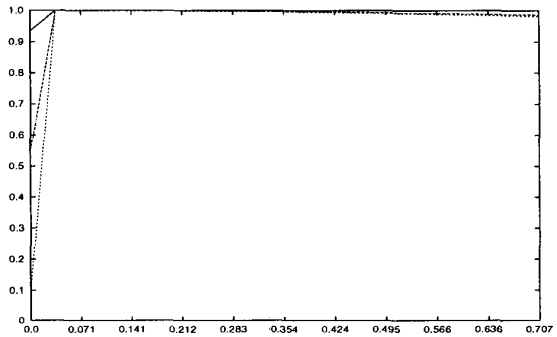

Figure 8: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, \alpha, 0.1, 0.1)$, absolute performance.

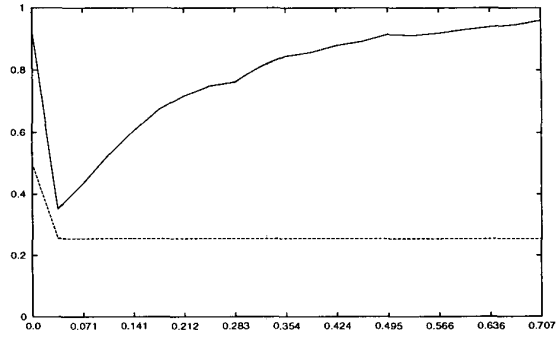

Figure 9: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, \alpha, 0.1, 0.1)$, relative performance.

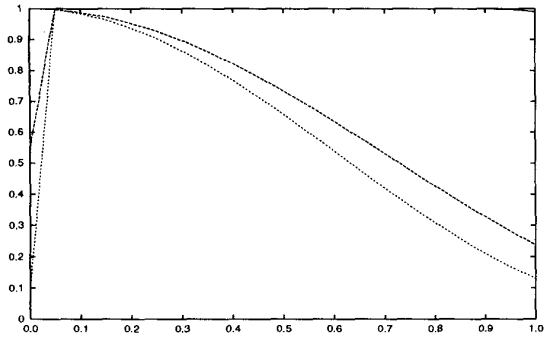

Figure 10: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, \beta, 0.1)$, absolute performance.

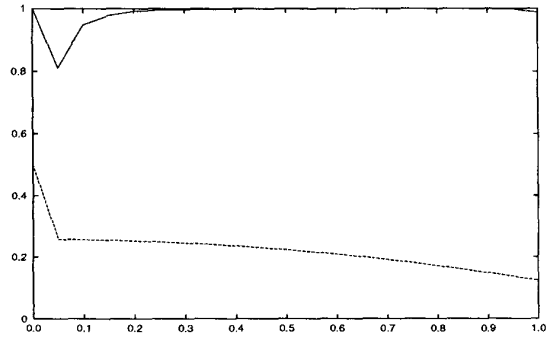

Figure 11: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, \beta, 0.1)$, relative performance.

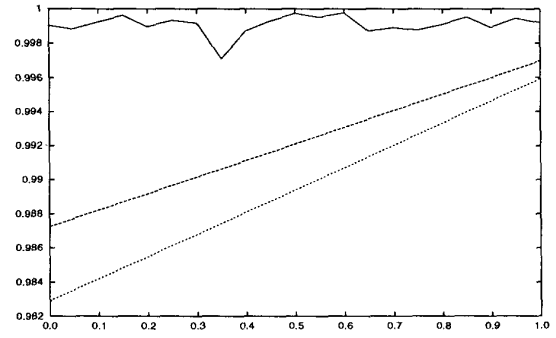

Figure 12: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, 0.1, \mu)$, absolute performance.

method on the $\mathcal{TCG}$ while β is varied between 0 and 1. Larger values of β decrease the size of rings; however, this feature seems not to influence the performance of the algorithm significantly.

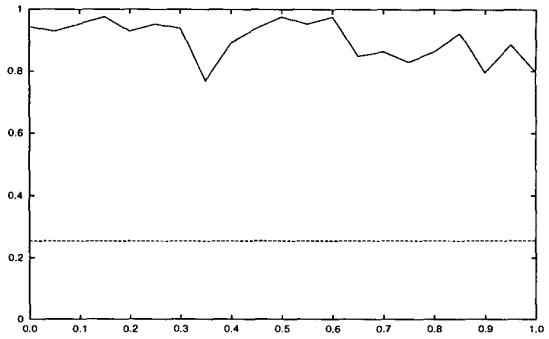

Figure 13: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, 0.1, \mu)$, relative performance.

Figures 12 and 13 display the absolute performance and relative performance respectively, of a static penalty method on the $\mathcal{TCG}$ while μ is varied between 0 and 1. Higher values of μ usually make the test case harder as the heights of the peaks are changed: local maxima are of almost the same height as the global one.

We therefore conclude, that the results obtained by an evolutionary algorithm using a static penalty approach depend mainly on the dimensionality n, the multimodality w, the connectedness α, and the ruggedness μ. On the other hand, they do not significantly seem to depend on the constraintness λ and the feasibility β.

It is interesting also to investigate the relationship between the value of penalty coefficient W and the quality of obtained solution. Figure 14 confirms another intuition connected with static penalties: it is difficult to guess the "right" value of the penalty coefficient as different landscapes require different values of W! Note that low values of W (i.e., values below 0.4) produce poor quality results; on the other hand, for larger values of W (i.e., values larger than 1.5), the quality of solutions drops slowly (it stabilizes later — for large W — at the level of 0.95). Thus the best values of penalty coefficient W for the particular landscape of the $\mathcal{TCG}(5, 10, 0.2, 0.9/\sqrt{5}, 0.5, 0.1)$ (which has $n^w = 10^5$ local optima) are in the range $[0.6, 0.75]$.

5 Summary

We have discussed briefly how the proposed test case generator $\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$ can be used for evaluation of a constraint-handling technique. As explained in section 3, the parameter n controls the dimensionality of the test function, the parameter w controls the modality of the function, the parameter λ controls the number of constraints in the search

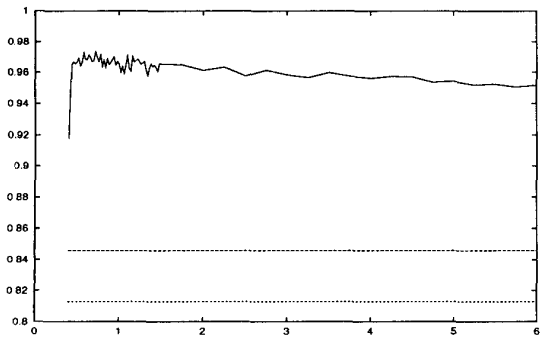

Figure 14: The quality of solution found for different values of penalty coefficient W.

space, the parameter α controls the connectedness of the feasible search space, the parameter β controls the ratio of the feasible to total search space, and the parameter μ controls the ruggedness of the test function.

We believe that such constrained test problem generator should serve the purpose of testing a constrained optimizer. In the previous section we have indicated how it can be used to evaluate merits and drawbacks of one particular constraint handling method (static penalties). Note that it is possible to analyse further the performance of a method by varying two or more parameters of the $\mathcal{TCG}$ (see [3] for a full discussion on these results).

The proposed test case generator is far from perfect. For example, it might be worthwhile to modify it further to parametrize the number of active constraints at the optima as well as equality constraints. It seems necessary to introduce a possibility of a gradual increment of peaks (in the current version of the test case generator, $w = 1$ implies one peak, and $w = 2$ implies 2^n peaks). Also, the difference between the lowest and the highest values of the peaks in the search space are, in the present model, too small. However, the proposed $\mathcal{TCG}$ is even in its current form an important tool for analysing any constraint-handling method (for any algorithm: not necessarily evolutionary algorithm), and it represents an important step in the "right" direction.

Acknowledgments

The second author acknowledges the support provided by the Alexander von Humboldt Foundation, Germany. The research reported in this paper was partially supported by the ESPRIT Project 20288 Cooperation Research in Information Technology (CRIT-2): "Evolutionary Real-time Optimization System for Ecological Power Control".

Bibliography

[1] Gregory, J. (1995). Nonlinear Programming FAQ, Usenet sci.answers. Available at ftp://rtfm.mit.edu/pub/usenet/sci.answers/nonlinear-programming-faq.

[2] Michalewicz, Z. (1995b). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics 1*(2), 177–206.

[3] Michalewicz, Z., K. Deb, M. Schmidt, and T. Stidsen (1999). Test-case Generator for Constrained Parameter Optimization Techniques, submitted for publication.

[4] Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the* 4^{th} *International Conference on Genetic Algorithms*, pp. 151–157. Morgan Kaufmann.

[5] Michalewicz, Z. and M. Schoenauer (1996). Evolutionary computation for Constrained Parameter Optimization Problems. Evolutionary Computation, Vol.4, No.1, pp.1–32.

[6] Orvosh, D. and L. Davis (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 650. Morgan Kaufmann.

[7] Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Proceedings of the* 3^{rd} *International Conference on Genetic Algorithms*, pp. 191–197. Morgan Kaufmann.

[8] van Kemenade, C.H.M. (1998). Recombinative evolutionary search. PhD Thesis, Leiden University, Netherlands, 1998.

[9] Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building better test functions. In L. Eshelmen (Editor), Proceedings of the 6th International Conference on Genetic Algorithms, Morgam Kaufmann, 1995.

[10] Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Evaluating evolutionary algorithms. Artificial Intelligence Journal, Vol.85, August 1996, pp.245–276.