The oracle penalty method

Martin Schlüter*, Matthias Gerdts†

* *Theoretical & Computational Optimization Group, University of Birmingham*
Birmingham B15 2TT, United Kingdom
{schluetm@maths.bham.ac.uk}

† *Department of Mathematics, University of Würzburg,*
Am Hubland, 97074 Würzburg, Germany
{gerdts@mathematik.uni-wuerzburg.de}

October 15, 2010

Abstract

A new and universal penalty method is introduced in this contribution. It is especially intended to be applied in stochastic metaheuristics like genetic algorithms, particle swarm optimization or ant colony optimization. The novelty of this method is, that it is an advanced approach that only requires one parameter to be tuned. Moreover this parameter, named *oracle*, is easy and intuitive to handle.

A pseudo-code implementation of the method is presented together with numerical results on a set of 60 constrained benchmark problems from the open literature. The results are compared with those obtained by common penalty methods, revealing the strength of the proposed approach.

Further results on three real-world applications are briefly discussed and fortify the practical usefulness and capability of the method.

Keywords: Constrained optimization, Global optimization, Penalty function, Stochastic metaheuristic, Ant colony optimization, MIDACO - Solver, Mixed integer nonlinear programming (MINLP).

1 Introduction

Constraints play an important role in the field of optimization and arise in many real-world applications. In general, a constrained optimization problem can be described as follows:

$$\begin{aligned} \text{Minimize} \quad & f(x), \\ \text{subject to:} \quad & g_i(x) \;=\; 0, \quad i \;=\; 1,...,m_{eq} \in \mathbb{N}, \\ & g_i(x) \;\geq\; 0, \quad i \;=\; m_{eq}+1,...,m \in \mathbb{N}, \end{aligned} \tag{1}$$

where $x = (x_1, ..., x_n)$ is the vector of decision variables of an $n \in \mathbb{N}$ dimensional search space (e.g. $\mathbb{R}^n$). The objective function $f(x)$ has to be minimized respectively to m_{eq} equality constraints $g_1, ..., g_{m_{eq}}$ and $m - m_{eq}$ inequality constraints $g_{m_{eq}+1}, ..., g_m$. It is to note that, alternatively, any equality constraint can be formulated by two inequality constraints.

Penalty methods are a well known technique to handle constrained optimization problems. Those functions transform a constrained problem into an unconstrained one by adding a penalty term to the original objective function. In particular, this approach to handle constraints is the most popular one among stochastic metaheuristics like genetic algorithms [15], particle swarm optimization [22], simulated annealing [23], scatter search [16] or ant colony optimization [6].

The big advantage of penalty methods is their simplicity and simple implementation. On the other side, simple penalty methods often perform very poorly on challenging constrained optimization problems, while more sophisticated ones normally require an additional tuning of many parameters to gain a sufficient performance. The burden of a good parameter selection for advanced penalty functions is a well known problem (see Coello Coello [3] or Yeniay [33]).

In this paper a conceptual new penalty method, named *oracle penalty method*, is presented. A slightly modified version of this method can be found in the previous works [29] and [30]. While no further explanations on the development of the methodology or the comparison with other penalty methods can be found there, those explanations and investigations are carried out in this contribution now.

The here proposed method is universal as it is applicable to any kind of optimization algorithm. While proper penalty functions for deterministic algorithms already exist and this method is of heuristic nature itself, it is especially intended to be employed in stochastic metaheuristics like the ones mentioned above. Moreover, the oracle penalty method aims at finding global optimal solutions, whereas several optimization runs might be required to adjust the one parameter required by the method. As stochastic metaheuristics normally also aim at finding global or nearly global solutions and often require several optimization runs due to their stochastic nature, the method seems very suitable for such kind of algorithms.

The name of the method is deduced from the predictive nature of its parameter, named oracle. This parameter directly corresponds to the global optimal (feasible) objective function value of a given problem and selecting an oracle parameter can therefore be seen as some kind of forecast. Even so there is no other parameter involved in the method than the oracle, it is still considered an advanced approach, absolutely competitive with other penalty functions commonly used in stochastic metaheuristics.

The paper is structured as follows: Firstly we analyze three common examples of penalty methods taken from the literature. Secondly, the key idea of the oracle method is developed introducing a basic version of the oracle penalty method. This basic version obviously lacks of robustness regarding the parameter selection. To strengthen the method regarding the parameter selection, three extensions for the basic version are carried out and explained in detail. These modifications finally lead to the extended oracle penalty function. An example of a pseudo-code implementation of the extended version together with a parameter update rule completes the discussion on the oracle penalty method. Thirdly, numerical results for 60 MINLP benchmark problems from the open literature are presented. Results obtained by the extended oracle penalty function are compared to those achieved by the penalty methods presented in section 1.1. Further results on three different real-world applications are presented. Finally, some conclusions are drawn.

1.1 Examples of common penalty methods

Three common penalty methods are presented and briefly analyzed now. The formulations presented here follow the concept of a *residual function* $res(x)$ used to measure the feasibility of an iterate x to problem (1). A residual function measures the constraint violations by applying a norm function over all m constraint violations of problem (1). This approach is commonly used and some explicit residual functions based on the l^1, l^2 and l^∞ norm are listed in Table 1. It is to note, that any feasible iterate x will correspond to a residual function value of zero.

Table 1: Examples of residual functions

Norm	residual function $res(x)$ for an iterate x
l^1	$res(x) = \sum_{i=1}^{m_{eq}} \|g_i(x)\| - \sum_{i=m_{eq}+1}^{m} \min\{0, g_i(x)\}$
l^2	$res(x) = \sqrt{\sum_{i=1}^{m_{eq}} \|g_i(x)\|^2 + \sum_{i=m_{eq}+1}^{m} \min\{0, g_i(x)\}^2}$
l^∞	$res(x) = \max\{ \|g_i(x)\|_{i=1,...,m_{eq}}, \|\min\{0, g_i(x)\}_{i=m_{eq}+1,...,m}\|\}$

Table 2 lists formulations of the *death*, *static* and *adaptive* penalty function $p(x)$ for an iterate x to problem (1) using a residual function $res(x)$. The last column of Table 2 contains the specific parameters, required by the corresponding penalty function.

Table 2: Examples of common penalty functions

Name	Penalty function $p(x)$ for an iterate x	Parameters
death	$p(x) = \begin{cases} f(x) & \text{, if } res(x) = 0 \\ \infty & \text{, if } res(x) > 0 \end{cases}$	none
static	$p(x) = f(x) + K \cdot res(x)$	K
adaptive	$p(x) = f(x) + \lambda(t) \cdot res(x)$ $\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t) & \text{, if case\#1} \\ \beta_2 \cdot \lambda(t) & \text{, if case\#2} \\ \lambda(t) & \text{, otherwise} \end{cases}$ case#1: The best individuals during the last k generations have been always feasible. case#2: The best individuals during the last k generations have been never feasible.	$\lambda(1),\ \beta_1,\ \beta_2,\ k$

The death penalty function is clearly the simplest penalty function possible. Any infeasible iterate will be penalized with infinity, while any feasible iterate is penalized with its objective function value. The main advantage of this method is the lack of any parameter, the main drawback is the inability to explore any infeasible region of the search space. Obviously this method can not be suitable for any challenging constrained optimization problem, where feasible iterates are difficult to find. Further information on this method can be found for example in Coit and Smith [4] or Michalewicz [26].

The static penalty function is a sum of the objective function and the residual function multiplied by the parameter K. This parameter is assumed to be quite large (e.g. 10^9) and enables the method to explore infeasible search regions. Even so this method seems to be already much more advanced than the death penalty function, it already comes with the drawback of one parameter to be selected. Further information on this method can be found for example in Homaifar et al. [20] or Kuri Morales and Villegas Quezada [25].

The adaptive penalty function is the sum of the objective function and the residual function multiplied by a dynamic factor $\lambda(t)$, which is updated for every generation t. Based on the progress of the algorithm in either finding feasible iterates (case#2) or improve feasible iterates (case#1), this factor increases or decreases the weight on the residual function in the penalty function. As this penalty function is able to dynamically adapt itself on the current progress of the algorithm, this approach seems suitable for challenging constrained problems. Nevertheless, requiring four parameters to be set in advance, this penalty function claims a lot of optimization effort itself. Further information on this method can be found for example in Hadj-Alouane and Bean [19] or Smith and Tate [32].

2 Oracle penalty method

In this section the oracle penalty method is described in detail. At first a basic version of the method is explained and modifications are developed which lead to an extended version. This extended version is robust enough to be applied on any general constrained optimization problem.

An example of an implementation together with an update rule for the oracle parameter complete this section.

2.1 Basic oracle penalty function

The key idea of the oracle penalty method is a transformation of the objective function $f(x)$ of problem (1) into an additional equality constraint $g_0(x) = f(x) - \Omega = 0$, where Ω is a parameter, named *oracle*. An objective function is redundant in the transformed problem definition and might be declared as a constant zero function $\tilde{f}(x)$. The transformed problem is then of the form:

$$
\begin{aligned}
\text{Minimize} \quad \tilde{f}(x) &\equiv 0 \\
\text{subject to:} \quad g_0(x) &= f(x) - \Omega = 0, \quad \Omega \in \mathbb{R}, \\
g_i(x) &= 0, \quad i = 1, ..., m_{eq} \in \mathbb{N}, \\
g_i(x) &\geq 0, \quad i = m_{eq}+1, ..., m \in \mathbb{N},
\end{aligned} \tag{2}
$$

Let now x^* denote the global optimal solution of problem (1). Then an oracle parameter $\Omega = f(x^*)$ would directly imply, that a feasible solution of problem (2) is the global optimal solution of problem (1).

Assuming that for a given optimization problem the optimal objective function value $f(x^*)$ is known, the problem definition (2) holds a significant advantage compared to definition (1). By transforming the objective function into an equality constraint, the current progress of the algorithm in minimizing the new constraint $g_0(x)$ and minimizing the residual of the original constraints $g_1(x), ..., g_m(x)$ becomes directly comparable. This comparability can be exploited by a penalty function, which balances its penalty weight on either the transformed objective function or the original constraints. The basic oracle penalty function (3) is an example of such a function:

$$p(x) = \alpha \cdot |f(x) - \Omega| + (1 - \alpha) \cdot res(x) \tag{3}$$

where α is given by:

$$
\alpha = \begin{cases} 1 - \dfrac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}} & , \text{ if } \quad res(x) \leq |f(x) - \Omega| \\[2ex] \dfrac{1}{2}\sqrt{\dfrac{|f(x)-\Omega|}{res(x)}} & , \text{ if } \quad res(x) > |f(x) - \Omega| \end{cases} \tag{4}
$$

The basic oracle penalty function (3) implicitly incorporates the transformed objective function $|f(x) - \Omega|$ and is therefore applicable to problem definition (1) without the necessity of the explicit problem transformation (2). It is to note, that the basic oracle penalty function assumes that $\Omega = f(x^*)$ and that the formulation is of heuristic nature.

The α factor is constructed as a dynamic weight between zero and one. This factor balances the penalty function value $p(x)$ in respect to the relationship between $|f(x) - \Omega|$ and $res(x)$. If $res(x) \leq |f(x) - \Omega|$ the quotient $\frac{|f(x)-\Omega|}{res(x)}$ will be greater or equal to one, which results in a value of α between 0.5 and 1. Hence, the penalty function will focus its weight on the transformed objective function. In case $res(x) > |f(x) - \Omega|$ the quotient $\frac{|f(x)-\Omega|}{res(x)}$ will be smaller than one, which results in a value of α between 0 and 0.5. Therefore the penalty function will focus its weight on the residual.

Figure 1 illustrates the basic oracle penalty function p(x) for an Ω parameter equal to zero according to objective function values $f(x) \in [-10, 10]$ and residual function values $res(x) \in [0, 10]$. It is to note, that the shape of the penalty function is not affected by different Ω parameters. A different Ω parameter will result only in a movement to the right ($\Omega > 0$) or left ($\Omega < 0$) according to the x-axis, representing the objective function values.

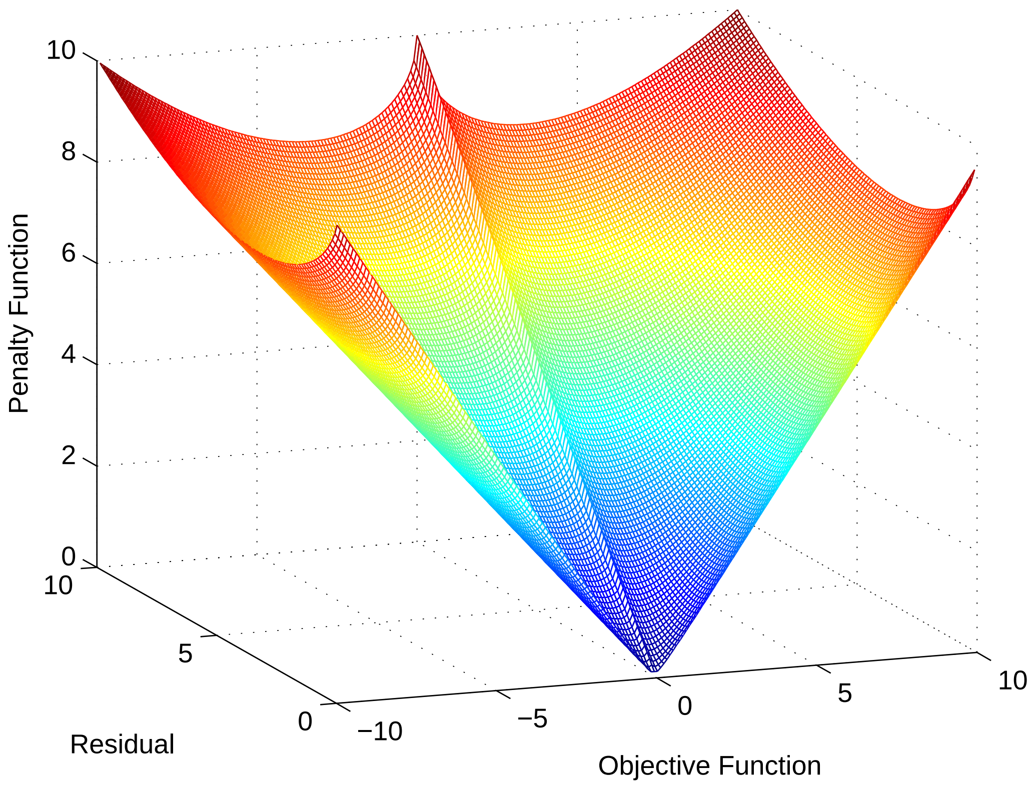

Figure 1: The basic oracle penalty function for $\Omega = 0$

The here proposed basic oracle penalty function suffers from a significant drawback. It is absolute sensitive with regard to the oracle parameter selection. To guide an algorithm to the global optimal solution of a problem, information about the global optimal objective function value is essential to apply the basic oracle penalty function.

2.2 Extensions for the basic oracle penalty function

With intention to apply the oracle penalty method on problems where no information is known about the optimal objective function value $f(x^*)$, this section describes modifications which make the method more robust regarding oracle parameter selection $\Omega \neq f(x^*)$. However, the modifications carried out here still assume two conditions for oracle parameters. This is $\Omega \geq f(x^*)$ and that at least one feasible solution $\tilde{x}$ exist, so that $\Omega = f(\tilde{x}) \geq f(x^*)$. These two conditions define a set of oracle parameters which is denoted as *trust oracles*. The set T_Ω defining all trust oracles is given by:

$$\mathrm{T}_\Omega \;:=\; \{f(\tilde{x}) \mid g_i(\tilde{x}) = 0 \ (i = 1, ..., m_{eq}) \ \wedge \ g_i(\tilde{x}) \geq 0 \ (i = m_{eq}+1, ..., m)\} \tag{5}$$

Obviously a trust oracle can also be used as oracle parameter within the basic oracle penalty function (3). Such a parameter selection will guide an algorithm, minimizing the corresponding penalty function, to a feasible solution $\tilde{x}$ with $f(\tilde{x}) = \Omega$. Nevertheless, based on the symmetric structure of the penalty function (3), no feasible iterate $\bar{x}$ with $f(\bar{x}) < \Omega$ will be penalized lower than $\tilde{x}$ with $f(\tilde{x}) = \Omega$.

The first modification concerns the desired property of the penalty function, to penalize any feasible iterate $\bar{x}$ with $f(\bar{x}) < \Omega$ lower than a feasible iterate $\tilde{x}$ with $f(\tilde{x}) = \Omega$ and therefore $p(\tilde{x}) = 0$. This can be easily achieved by splitting the penalty function (3) into two cases. The first case affects any iterate with an objective function value greater than the oracle or any infeasible iterate, while the second case concerns only feasible iterates with an objective function value lower than the oracle:

$$p(x) \;=\; \begin{cases} \alpha \cdot |f(x) - \Omega| + (1-\alpha)\cdot res(x) & , \text{ if } \quad f(x) > \Omega \quad \text{or} \quad res(x) > 0 \\ \\ -|f(x) - \Omega| & , \text{ if } \quad f(x) \le \Omega \quad \text{and} \quad res(x) = 0 \end{cases} \tag{6}$$

Due to this modification any feasible iterate $\bar{x}$ with $f(\bar{x}) < \Omega$ will be penalized with a negative value. Moreover, a lower $f(\bar{x})$ directly results in a better (lower) penalty function value, which seems quite reasonable.

The second modification concerns infeasible iterates corresponding to objective function values lower than the oracle parameter Ω. Imagine a just slightly infeasible iterate $\hat{x}$ with an objective function value $f(\hat{x})$ much lower than Ω. Such an iterate would be penalized higher than any iterate with the same residual greater than $f(\hat{x})$ and lower than Ω. Modifying the α factor by adding a new case overcomes this undesired property. In case of an infeasible iterate $\hat{x}$ with $f(\hat{x}) \le \Omega$, the penalty function should penalize the iterate with its residual $res(\hat{x})$. This means, α is zero in such a case:

$$\alpha \;=\; \begin{cases} 1 - \dfrac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}} & , \text{ if } \quad res(x) \le |f(x) - \Omega| \quad \text{and} \quad f(x) > \Omega \\ \\ \frac{1}{2}\sqrt{\frac{|f(x)-\Omega|}{res(x)}} & , \text{ if } \quad res(x) > |f(x) - \Omega| \quad \text{and} \quad f(x) > \Omega \\ \\ 0 & , \text{ if } \quad f(x) \le \Omega \end{cases} \tag{7}$$

The third modification concerns iterates $\dot{x}$ with $f(\dot{x}) > \Omega$ and $res(\dot{x}) \le \frac{|f(\dot{x})-\Omega|}{3}$. For such iterates a highly undesired effect takes place: A iterate $\dot{x}$ with $f(\dot{x}) > \Omega$ and $res(\dot{x}) < \frac{|f(\dot{x})-\Omega|}{3}$ will be penalized higher than an iterate $\ddot{x}$ with $f(\dot{x}) = f(\ddot{x})$ and $res(\ddot{x}) = \frac{|f(\ddot{x})-\Omega|}{3}$. In other words, even so the iterate $\dot{x}$ is equally good in the objective function as the iterate $\ddot{x}$ and $\dot{x}$ has a lower residual (maybe even zero) than $\ddot{x}$, it will be penalized higher than $\ddot{x}$.

This effect is caused by the construction of α and can also be observed in the shape of the basic penalty function in Figure 1. The area of the shape which bends upwards in the front right part of the figure relates to this effect. Now a modification is carried out, which resolves this effect by constructing an additional α case. This α case will ensure, that any iterate $\dot{x}$ with the above properties is penalized with the same value as an iterate $\ddot{x}$ with the above properties. This means, that the mentioned area in Figure 1 would be plane (see Figure 2).

To obtain this additional α case it is shown first that the penalty function $p(x)$ in (6), concerning only iterates x with an identical objective function value $f(x) > \Omega$, takes its minimum for an iterate with $res(x) = \frac{|f(x)-\Omega|}{3}$. Then the corresponding penalty function value will be calculated and used to deduce the additional α case.

Let

$$f(x) > \Omega \quad \text{and} \quad res(x) \le |f(x) - \Omega|$$

then

$$\alpha = 1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}}$$

and

$$p(x) = \alpha \cdot |f(x) - \Omega| + (1 - \alpha) \cdot res(x)$$

$$\Longrightarrow \quad p(x) = (1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}}) \cdot |f(x) - \Omega| + (1 - (1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}})) \cdot res(x)$$

$$= |f(x) - \Omega| - \frac{|f(x) - \Omega|}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}} + \frac{res(x)}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}}$$

$$= |f(x) - \Omega| - \frac{|f(x) - \Omega|\sqrt{res(x)}}{2\sqrt{|f(x) - \Omega|}} + \frac{res(x)\sqrt{res(x)}}{2\sqrt{|f(x) - \Omega|}}$$

To investigate the deviation of $p(x)$ in respect to $res(x)$, the residual function $res(x)$ is substituted by y and the function $\tilde{p}(y)$ is defined by:

$$\tilde{p}(y) = (1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{y}}}) \cdot |f(x) - \Omega| + (1 - (1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{y}}})) \cdot y$$

The deviation of $\tilde{p}(y)$ in respect to y is given by:

$$\frac{d}{dy}\tilde{p}(y) = -\frac{|f(x) - \Omega|}{4\sqrt{|f(x) - \Omega|}\sqrt{y}} + \frac{3\sqrt{y}}{4\sqrt{|f(x) - \Omega|}}$$

Let

$$\frac{d}{dy}\tilde{p}(y) = 0$$

then

$$\frac{|f(x) - \Omega|}{4\sqrt{|f(x) - \Omega|}\sqrt{y}} = \frac{3\sqrt{y}}{4\sqrt{|f(x) - \Omega|}}$$

$$\Longleftrightarrow \quad \frac{|f(x) - \Omega|}{\sqrt{y}} = 3\sqrt{y}$$

$$\Longleftrightarrow \quad y = \frac{|f(x) - \Omega|}{3}$$

Under the above assumption the second deviation of $\tilde{p}(y)$ with respect to y is given by:

$$\frac{d^2}{d^2y}\tilde{p}(y) = \frac{\sqrt{|f(x) - \Omega|}}{8y\sqrt{y}} + \frac{3}{8\sqrt{y}\sqrt{|f(x) - \Omega|}} \ > \ 0$$

This means that the penalty function (under the above assumptions) takes its minimum for iterates x with identical objective function value $f(x)$ and $res(x) = \frac{|f(x)-\Omega|}{3}$. Now the penalty function value for such an iterate is calculated.

Let

$$f(x) > \Omega \quad \text{and} \quad res(x) = \frac{|f(x) - \Omega|}{3}$$

then

$$\alpha = 1 - \frac{1}{2\sqrt{3}}$$

$$\implies \quad p(x) = (1 - \frac{1}{2\sqrt{3}}) \cdot |f(x) - \Omega| + \frac{1}{2\sqrt{3}} \cdot \frac{|f(x) - \Omega|}{3}$$

$$= |f(x) - \Omega| - \frac{|f(x) - \Omega|}{2\sqrt{3}} + \frac{|f(x) - \Omega|}{6\sqrt{3}}$$

$$= |f(x) - \Omega| \cdot (1 - \frac{1}{2\sqrt{3}} + \frac{1}{6\sqrt{3}})$$

$$= |f(x) - \Omega| \cdot \frac{6\sqrt{3} - 2}{6\sqrt{3}}$$

Now the penalty function (6) (under the above assumptions) is set equal to the above optimal penalty function value to deduce α.

Let

$$\alpha \cdot |f(x) - \Omega| + (1 - \alpha) \cdot res(x) = |f(x) - \Omega| \cdot \frac{6\sqrt{3} - 2}{6\sqrt{3}}$$

then

$$\alpha \cdot (|f(x) - \Omega| - res(x)) + res(x) = |f(x) - \Omega| \cdot \frac{6\sqrt{3} - 2}{6\sqrt{3}}$$

$$\implies \quad \alpha \cdot (|f(x) - \Omega| - res(x)) = |f(x) - \Omega| \cdot \frac{6\sqrt{3} - 2}{6\sqrt{3}} - res(x)$$

$$\implies \quad \alpha = \frac{|f(x) - \Omega| \cdot \frac{6\sqrt{3}-2}{6\sqrt{3}} - res(x)}{|f(x) - \Omega| - res(x)}$$

This α factor can now be applied as additional case within (7) for iterates x with $res(x) \leq |f(x) - \Omega|$ and $f(x) > \Omega$. All iterates x with identical objective function value $f(x) > \Omega$ and $res(x) \leq \frac{|f(x)-\Omega|}{3}$ will then be penalized with $|f(x) - \Omega| \cdot \frac{6\sqrt{3}-2}{6\sqrt{3}}$.

2.3 Extended oracle penalty function

In this section the basic oracle penalty function, presented in section 2.1, is extended by the three modification explained in section 2.2. The extended oracle penalty function is of the form:

$$p(x) = \begin{cases} \alpha \cdot |f(x) - \Omega| + (1 - \alpha) \cdot res(x) & , \text{ if } \; f(x) > \Omega \; \text{ or } \; res(x) > 0 \\ -|f(x) - \Omega| & , \text{ if } \; f(x) \leq \Omega \; \text{ and } \; res(x) = 0 \end{cases} \tag{8}$$

where α is given by:

$$\alpha = \begin{cases} \frac{|f(x)-\Omega| \cdot \frac{6\sqrt{3}-2}{6\sqrt{3}} - res(x)}{|f(x)-\Omega| - res(x)} & \text{, if } f(x) > \Omega \text{ and } res(x) < \frac{|f(x)-\Omega|}{3} \\ 1 - \frac{1}{2\sqrt{\frac{|f(x)-\Omega|}{res(x)}}} & \text{, if } f(x) > \Omega \text{ and } \frac{|f(x)-\Omega|}{3} \leq res(x) \leq |f(x)-\Omega| \\ \frac{1}{2}\sqrt{\frac{|f(x)-\Omega|}{res(x)}} & \text{, if } f(x) > \Omega \text{ and } res(x) > |f(x)-\Omega| \\ 0 & \text{, if } f(x) \leq \Omega \end{cases} \tag{9}$$

Figure 2 illustrates the extended oracle penalty function $p(x)$ for an Ω parameter equal to zero according to objective function values $f(x) \in [-10, 10]$ and residual function values $res(x) \in [0, 10]$. Again, it is to note, that the shape of the penalty function itself is not affected by different choices of the oracle parameter. Those will only result in a movement of the shape to the right ($\Omega > 0$) or the left ($\Omega < 0$).

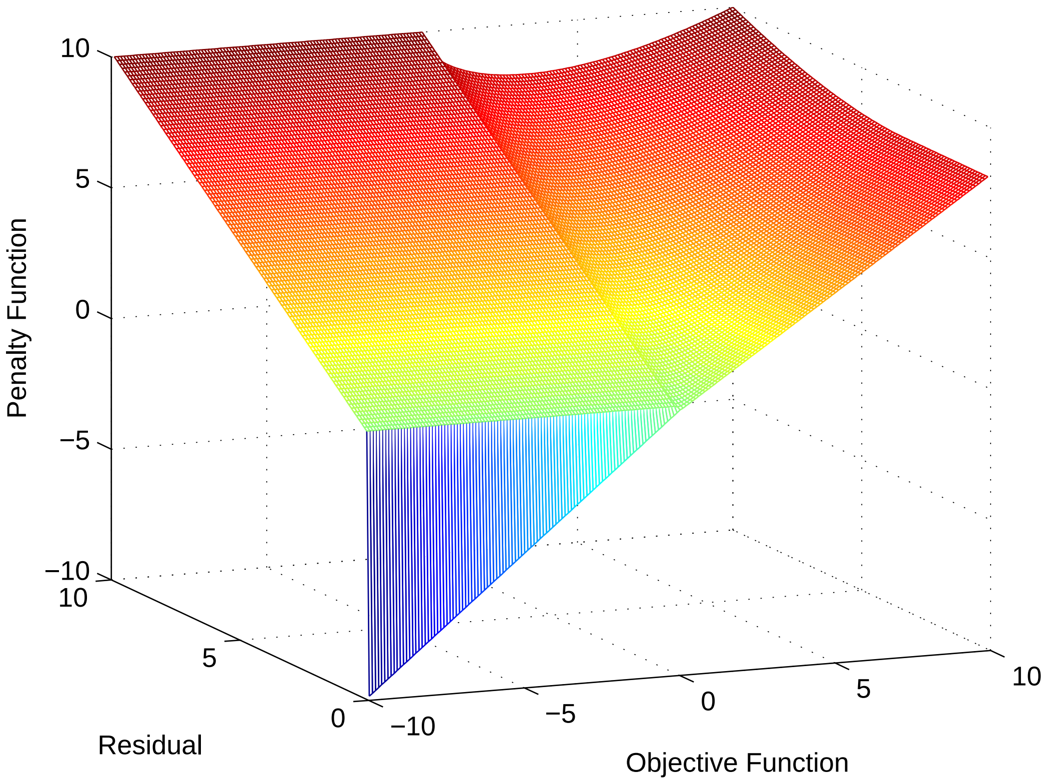

Figure 2: The extended oracle penalty function for $\Omega = 0$

Now it is explained how the three modifications can be observed in the shape of the extended oracle penalty function, illustrated in Figure 2. The first modification, concerning feasible iterates $\bar{x}$ with $f(\bar{x}) < \Omega$, corresponds to the vertical triangular (in the left front), penalizing those iterates with a negative penalty function value. The second modification, concerning infeasible iterates $\hat{x}$ with $f(\hat{x}) \leq \Omega$, corresponds to the plane area (in the left back), penalizing those iterates with their residual function value. The third modification, concerning iterates $\dot{x}$ with $f(\dot{x}) > \Omega$ and $res(\dot{x}) \leq \frac{|f(\dot{x})-\Omega|}{3}$, corresponds to the small plane area (in the right front) of the shape of the penalty function. While this area is nonlinear in the basic oracle penalty function shape, it is now plane, meaning that those iterates are penalized equal as iterates $\ddot{x}$ with $f(\ddot{x}) = f(\dot{x})$ and $res(\ddot{x}) = \frac{|f(\ddot{x})-\Omega|}{3}$.

As explained in section 2.4 those modifications are intended to make the method robust regarding trust oracles (5). However, due to the first and second modification, the extended oracle penalty function can also be applied for sufficiently large oracle parameters. Sufficiently large means here that $\Omega > f(x)$ for any iterate x. For such an oracle parameter only the first and second modifications are relevant in the penalty function. This means, infeasible iterates x will be penalized with their residual $res(x) \geq 0$, while feasible iterates x will be penalized with the negative distance $-|f(x) - \Omega|$. Hence, for sufficiently large oracle parameters the extended oracle penalty function will act very similar to a static penalty function (see Table 2).

2.4 Update rule and implementation

Here a simple but effective update rule for the oracle parameter Ω is presented. It is intended to be applied if no information about the optimal feasible objective function value is known and several optimization runs are performed. **However, it is not assumed that the oracle parameter Ω is changed during an optimization run!** Updating the oracle parameter during an optimization run implies two major failures. First, imagine a case of an ambitious oracle parameter selection (e.g. Ω=100) and the current optimization run reveals at first a feasible solution with higher objective than Ω (e.g. $f(x) = 150$) and an oracle update (e.g. Ω =150) takes place, the penalty function will then behave like a static penalty, not allowing the algorithm to explore infeasible regions with corresponding objective to the original oracle (e.g. Ω=100). In other words, the main idea of the oracle penalty method is corrupted, if preliminary updates of the oracle parameter are undertaken during an optimization run. Second, changing the oracle parameter Ω during an optimization run corrupts the reference system of penalty values created previously in the same run. In other words, if the penalty of an iterate was at the beginning of an optimization run calculated as p^1 based on an oracle parameter Ω^1, the same iterate would have another penalty value p^2 based on the newly updated oracle parameter Ω^2. Hence, the reference system of previously calculated penalty values got lost. **Oracle parameter updates should only take place, when a single optimization run is finished and a new one is started.**

Let Ω^i denote the oracle parameter used for the i-th optimization run. Furthermore f^i and res^i should denote the objective function value and residual function value obtained by the i-th optimization run. The here proposed update rule will initialize the oracle parameter Ω^1 for the very first run with a sufficiently large parameter. This means $\Omega > f(x)$ for any iterate x to a given problem. As explained in section 2.3 the extended oracle penalty function will than act very similar to a static penalty function. This means the method is focused completely on the residual until a feasible solution is found. Please note, that a too large initialization of the oracle parameter (e.g. $\Omega = 10^{32}$) can cause numerical problems. An initialization of the oracle parameter of about $\Omega = 10^6$ or $\Omega = 10^9$ is therefore recommended for most applications.

The oracle parameters $\Omega^{2,3,\cdots}$ used for any further optimization run should then be calculated by the following update rule:

$$\Omega^i = \begin{cases} f^{i-1} & \text{,if } f^{i-1} < \Omega^{i-1} \quad \text{and} \quad res^{i-1} = 0 \\ \Omega^{i-1} & \text{,else} \end{cases} \tag{10}$$

According to (10) the oracle parameter Ω^i for the i-th optimization run is then always equal to the lowest known feasible objective function value or (in case no feasible solution is found so far) remains sufficiently large, until a feasible solution is found. This is done by updating the oracle parameter with the latest feasible solution which has a lower objective function value than the present oracle parameter or leaving the oracle parameter unaffected, in case the solution is infeasible or has a larger objective function value than the present oracle parameter.

It is to note, that for a specific problem an intuitive initialization of the very first oracle parameter Ω^1 by the user is possible as well (see Section 3.2.1 for an example). Imagine a real world application with an already known (feasible) solution, in such a case the user could initialize Ω^1 with a value reasonable lower than the current known solution objective function value. This property of an easy and intuitive handling of the oracle method is seen as quite appealing for practitioners.

Algorithm 1 gives a pseudo-code implementation of the extended oracle penalty function. For a given objective function value $f(x)$, a residual value $res(x)$, an oracle Ω and some tolerance $acc \geq 0$, this algorithm calculates the corresponding penalty function value $p(x)$. Due to the `if`-clauses in this implementation, the computational expensive α parameters are only calculated case depended and if necessary.

Algorithm 1 `Extended oracle penalty function`

if $f(x) \leq \Omega$ **and** $res(x) \leq acc$ **then**

  $p(x) = f(x) - \Omega$

else

  if $f(x) \leq \Omega$ **then**

    $p(x) = res(x)$

  else

    if $res(x) < \frac{f(x)-\Omega}{3}$ **then**

      $\alpha = \frac{(f(x)-\Omega)\frac{6\sqrt{3}-2}{6\sqrt{3}} - res(x)}{f(x)-\Omega-res(x)}$

    end if

    if $res(x) \geq \frac{f(x)-\Omega}{3}$ **and** $res(x) \leq (f(x) - \Omega)$ **then**

      $\alpha = 1 - \frac{1}{2\sqrt{\frac{f(x)-\Omega}{res(x)}}}$

    end if

    if $res(x) > (f(x) - \Omega)$ **then**

      $\alpha = \frac{1}{2}\sqrt{\frac{f(x)-\Omega}{res(x)}}$

    end if

    $p(x) = \alpha(f(x) - \Omega) + (1 - \alpha)res(x)$

  end if

end if

return $p(x)$

Implementations of the extended oracle penalty method in the programming languages `Fortran`, `C/C++` and `Matlab` can be found online [31] and can freely be downloaded at

Http://www.midaco-solver.com/oracle.html.

3 Numerical Results

To evaluate the performance of the oracle method and compare it with common penalty functions, numerical results are presented and analyzed in this section. Only the extended oracle penalty function is considered here, as the basic version obviously lacks of practical relevance. The three examples of common penalty functions presented in Table 2 are used here as concurrent methods.

As stochastic metaheuristic we consider `MIDACO` [31], which is an advanced implementation of an ant colony optimization (ACO) algorithm for mixed integer search domains. The theoretical fundamentals of the algorithm are described in [29] and [30]. For the numerical results presented in this section, we employed different penalty functions within `MIDACO`.

The ACO metaheuristic for continuous [28] and its extension to mixed integer search domains [30] is based on multi-kernel probability density functions (PDF's). In case of `MIDACO` Gauss distributions are considered. In every generation some solution candidates (called *ants*) for the optimization problem are generated successively by sampling random numbers according to those PDF's in every dimension of the search space. By saving and ranking the ants in some solution archive based on their fitness, parameters for the PDF's like mean, deviation and kernel weights are adjusted and influence the creation of ants in the following generation. Here the fitness of ants refers to either their corresponding objective function value (in case of unconstrained problems) or to some penalty function value (in case of constrained problems).

As this metaheuristic is of very general nature, we feel that the comparative results of different penalty functions tested within this framework are transferable to other metaheuristics, which also rank their individuals exclusively by some fitness function. Moreover, as described in section 2.1 the oracle penalty method aims on a problem transformation that happens outside the optimization algorithm and no algorithmic interaction (except passing the objective function, residual and penalty values) happens between the penalty function and the metaheuristic. Therefore we believe that the following results are of representative nature.

In the following we present numerical results on a set of 60 constrained benchmark problems from the open literature and we briefly discuss some existing results of the oracle penalty method for real-world applications.

3.1 Numerical results on 60 constrained benchmark problems

A set of 60 constrained benchmark problems from the open literature is considered to compare the performance of different penalty functions. Details on all benchmark problems can be found in the Appendix in Table 15.

As penalty functions we consider besides the extended oracle penalty function the static, death and adaptive one (see Table 2). For the numerical test different parameter setups for some penalty functions have been applied. Table 3 contains information on the penalty functions and parameters used. The extended oracle penalty function was tested with two setups. One time the oracle parameter remained constant throughout all test runs for a problem, while the other time this parameter was updated according to the update rule presented in Section 2.4. The static penalty was tested with only one setup and the death penalty does not require any parameter. The adaptive penalty was tested with three different parameter setups, where the first one (adaptive$_1$) uses the same parameters as proposed in Coello Coello [3].

Table 3: Penalty functions and their parameters considered for numerical results

Abbreviation	Penalty function	Section	Parameters
oracle$_{update}$	Extended oracle	2.3, 2.4	$\Omega^1 = 10^9$, $\Omega^{2,3,\cdots}$ updated according to equation (10)
oracle$_{fix}$	Extended oracle	2.3	$\Omega = 10^9$
static	Static	1.1	$K = 10^9$
death	Death	1.1	none
adaptive$_1$	Adaptive	1.1	$\lambda(1) = 100$, $\beta_1 = 1$, $\beta_2 = 2$, $k = 20$
adaptive$_2$	Adaptive	1.1	$\lambda(1) = 50$, $\beta_1 = 1.5$, $\beta_2 = 2.5$, $k = 10$
adaptive$_3$	Adaptive	1.1	$\lambda(1) = 200$, $\beta_1 = 2$, $\beta_2 = 3$, $k = 40$

Every problem of the set was tested 100 times with a different random seed for the random number generator within `MIDACO`. For every single test run two stopping criterions were applied. The first one is a maximal budget of fitness evaluations, where one fitness evaluation equals an objective function evaluation and all constraint function evaluations. We assigned a budget of $10000 \cdot n$ fitness evaluations for every test run, where n is the dimension of the optimization variables. The second one is a success criteria based on the best known objective function value $f(x^*)$ (see Table

15). If a feasible solution x with an objective function value $f(x)$ was found, so that:

$$\frac{|f(x) - f(x^*)|}{f(x^*)} \leq acc, \tag{11}$$

the run was stopped and recorded as successful in finding the global optimal solution. Here an accuracy acc of 10^{-4} was used, which was also set as accuracy for the constraints (see acc tolerance in Algorithm 1).

The overall performance of all tested penalty functions on the set of 60 problems is displayed in Table 5, where Table 4 explains the abbreviations used. Detailed results on every problem and every penalty function can be found in the Appendix in Table 17, where Table 16 explains the abbreviations used.

Table 4: Abbreviations for Table 5

Abbreviation	Explanation
Penalty	Penalty function used in `MIDACO`
Optimal (out of 60, [%])	Total number of problems where at least one out of 100 runs a **global** optimal solution could be found by the corresponding penalty function.
Feasible (out of 60, [%])	Total number of problems where at least one out of 100 runs a feasible solution could be found by the corresponding penalty function.
% Feasible (overall)	Percentage of all test runs by the corresponding penalty function in which a feasible solution was found.
% Optimal (overall)	Percentage of all test runs by the corresponding penalty function in which a **global** optimal solution was found.

Table 5: Overall performance of different penalty methods

Penalty	Optimal (out of 60, [%])	Feasible (out of 60, [%])	% Feasible (overall)	% Optimal (overall)
Oracle_{update}	56 [0.933]	60 [1.000]	0.919	0.733
Oracle_{fix}	48 [0.800]	60 [1.000]	0.995	0.694
Static	50 [0.833]	60 [1.000]	0.997	0.693
Death	40 [0.667]	57 [0.950]	0.831	0.542
Adaptive_1	48 [0.800]	59 [0.983]	0.903	0.688
Adaptive_2	49 [0.817]	59 [0.983]	0.901	0.699
Adaptive_3	52 [0.867]	60 [1.000]	0.907	0.703

With 56 out of 60 problems the extended oracle penalty function with updated oracles had the highest potential in solving a problem to the global optimum. With 52 out of 60 problems the Adaptive_3 performed second best in finding global optimal solutions. Interpreting this result, one has to take into account, that the adaptive penalty function needs four parameters to be tuned, while the oracle penalty method only needs one.

As expected, the extended oracle penalty function with fixed oracle and the static penalty function performed very similar (see Section 2.3), this can especially be observed in the statistics on all runs. Those two penalty functions performed most robust in finding feasible solutions, but lacked of potential to find global optimal solutions on more difficult problems. Not surprisingly the death penalty performed worst in all categories.

That the death penalty was able to locate the global optimal solution in 40 out of 60 cases, means that two third of the test problems are trivial or easy. Nevertheless, this leaves 20 non-trivial problems in the set on which significant differences between the tested penalty functions could be observed. On four problems no penalty function was able to find the global optimal solution. On these problems the results are not clear. On the problems nvs02 and nvs05 the Oracle_{update} performed best, while on floudas4 and ST_E36 the Oracle_{fix} and static penalty function performed best (see Appendix).

3.2 Numerical results on real-world applications

Here we briefly discuss some results on real-world applications obtained by the oracle penalty method. The first application, the *Tennessee Eastman Process* [7], is a well known case study in chemical engineering. The second application is a distillation column sequencing model taken from Floudas and Pardalos book *Collection of Test Problems for Constrained Global Optimization Algorithms* [14]. The third application is an optimal control problem of an aircraft manoeuvre introduced by Kaya and Noakes [21] to which several comparative results can be found in the literature.

3.2.1 Tennessee Eastman Process

The results on the *Tennessee Eastman Process* (TEP) presented in this subsection are taken from [30] and can be seen as an example of the successful application of the oracle penalty method (used within a stochastic metaheuristic) on a complex real-world application.

The TEP was introduced by Downs and Vogel [7] and is since then widely used in the literature as challenging benchmark. The TEP simulates a chemical plant where the objective is to minimize the operation cost. In [30] a MINLP formulation is considered which incorporates 171 DAEs (30 ODEs and 141 algebraic equations). The variable and constraint dimensions of the MINLP are shown in Table 6.

Table 6: MINLP problem dimensions for the TEP

Dimension	Value	Explanation
n	43	Number of variables in total
n_{int}	7	Number of integer variables
m	11	Number of constraints in total
m_{eq}	1	Number of equality constraints

Three different global optimization solvers have been tested and compared on this MINLP formulation of the TEP in [30]: SSM [9] (a scatter search algorithm), MITS [11] (a tabu search algorithm) and ACOmi [30] (an ant colony optimization algorithm).

In case of ACOmi the oracle penalty method was used to handle the constraints. For ACOmi two different setups where considered: one with a reasonable oracle parameter $\Omega = 100$ and one with a very large oracle parameter $\Omega = 10^{12}$. Please note that the (feasible) initial point for all solvers had an objective function value of 159.33. Therefore the first ACOmi setup with $\Omega = 100$ can be seen as a reasonable oracle choice.

Table 8 compares the results of all solvers for 10 test runs, where Table 7 gives the abbreviations used.

Table 7: Abbreviations for Table 8

Abbreviation	Explanation
Solver	Solver used for corresponding line of results
f_{best}	Best objective function value out of all runs
f_{worst}	Worst objective function value out of all runs
f_{mean}	Mean objective function value out of all runs
eval_{mean}	Mean number of function evaluations out of all runs
time_{mean}	Mean time out of all runs

Table 8: Results for the TEP

Solver	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
$\text{ACOmi}_{\Omega=100}$	84.19	152.51	112.65	10636	6593.59
$\text{ACOmi}_{\Omega=10^{12}}$	147.57	148.72	148.06	10113	9843.04
SSM	147.547	148.82	147.991	21822	10857.8
MITS	147.951	149.015	148.484	19309	10435.8

It can be seen, that only ACOmi in its setup with a reasonable oracle parameter choice of $\Omega = 100$ was able to find a significantly better solution. As the second ACOmi setup with $\Omega = 10^{12}$ (which leads to an oracle penalty function behavior alike those of a static penalty function) performed very similar to SSM and MITS, it is shown that not ACOmi but the oracle penalty method and its selected oracle parameter played the key role.

3.2.2 Distillation column sequencing test problems

In this subsection two constrained global optimization test problems taken from Floudas and Pardalos [14] (Chapter 5.5 Test Problem 4 and Chapter 5.6 Test Problem 5) are considered. In the following we refer to them as TP4 and TP5. These applications simulate some chemical component separation by distillation columns as MINLP problems. Table 9 and Table 10 lists the MINLP problem dimensions for TP4 and TP5.

Table 9: MINLP problem dimensions for TP4

Dimension	Value	Explanation
n	52	Number of variables in total
n_{int}	2	Number of integer variables
m	38	Number of constraints in total
m_{eq}	35	Number of equality constraints

Table 10: MINLP problem dimensions for TP5

Dimension	Value	Explanation
n	113	Number of variables in total
n_{int}	3	Number of integer variables
m	71	Number of constraints in total
m_{eq}	67	Number of equality constraints

For both applications best known solutions are reported in [14]. `MIDACO` [31] (which incorporates the oracle penalty method with automated restarts and oracle updates) was used to optimize these applications. Table 11 compares the reported best known solutions in the literature and those obtained by `MIDACO` using the oracle penalty method.

Table 11: Comparison of solutions (objective function values) for TP4 and TP5

Problem	best known solution reported in [14]	best known solution found by `MIDACO`
TP4	0.626	0.337
TP5	2.579	0.316

In both cases a significant better solution could be obtained by `MIDACO` using the oracle penalty method. Implementations (in Matlab) of the TP4 and TP5 applications together with the here mentioned `MIDACO` solutions can freely by downloaded from [31] at

Http://www.midaco-solver.com/applications.html

for verification purposes.

3.2.3 F-8 aircraft control problem

In this subsection an optimal control of an aircraft manoeuvre is discussed. This application is known as the F-8 aircraft control problem introduced by Kaya and Noakes [21]. Here we refer to a formulation of this application that can be downloaded from *mintOC* [27] at

Http://mintoc.de/index.php/F-8_ aircraft.

Several reference solutions to this application can be found at this online reference. Among those are solutions obtained by well known solvers such as *BONMIN* and *IPOPT*. Please note, that none of the mentioned two solvers is capable to solve this application to its current best known solution. Hence we consider this as a challenging application with good possibilities to compare the solution quality.

MIDACO [31] (which incorporates the oracle penalty method with automated restarts and oracle updates) was used to optimize this application. To apply MIDACO on the F-8 aircraft control problem we transformed it into a NLP black-box model. The objective of the model was the final time of the aircraft manoeuvre. The constraints of the model incorporated the integration over the states of the original control problem. Furthermore three additional constraints assured a final state condition. In total this lead to an amount of 183 constraints in the black-box model. Six different stages to simulate the integer control were considered, which resulted in six continuous optimization variables. The NLP black-box model dimensions are listed in Table 12.

Table 12: NLP black-box model dimensions for F-8 aircraft control problem

Dimension	Value	Explanation
n	6	Number of variables
m	183	Number of constraints in total
m_{eq}	3	Number of equality constraints

In the F-8 aircraft model the final state constraints hold that all three differential states must be zero at the end of the manoeuvre. These equality constraints are highly sensitive to the precision of the six continuous variables in the model. As MIDACO is a stochastic solver a moderate precision of 10^{-2} is assumed for the constraint violation in the l^{∞}-Norm. This precision is a crucial factor because MIDACO employees the oracle update rule described in Section 2.4 which assumes only feasible solutions as valid update candidates. Therefore a moderate precision enables the algorithm to realize more oracle updates in shorter computation time.

For the optimization run on the F-8 aircraft model we assigned a time budget of 8 hours on a PC with 2 GHz clock rate and 2 GB RAM working memory. In this 8 hours 304605 black-box model evaluations were performed, where not every evaluation necessarily implied an integration over the differential states. In Table the currently best know solution by Sager [27] is compared to the one obtained by MIDACO using the format taken from

Http://mintoc.de/index.php/F-8_ aircraft.

Table 13: F-8 aircaft control problem solutions

Arc	w(t)	Sager	MIDACO
1	1	1.13492	1.142628
2	0	0.34703	0.408936
3	1	1.60721	1.476987
4	0	0.69169	0.488491
5	1	0	0.000017
6	0	0	0.225701
Infeasibility	-	2.21723e-07	9.998e-03
Objective	-	3.78086	3.742759

The differential states corresponding to the MIDACO solution are displayed in Figure 3. From both, the numerical solutions and the differential states, it can be seen that the MIDACO solution is very close to the Sager solution. The remaining differences and slightly better objective function value finds its explanation in the moderate precision in the infeasibility of the MIDACO solution.

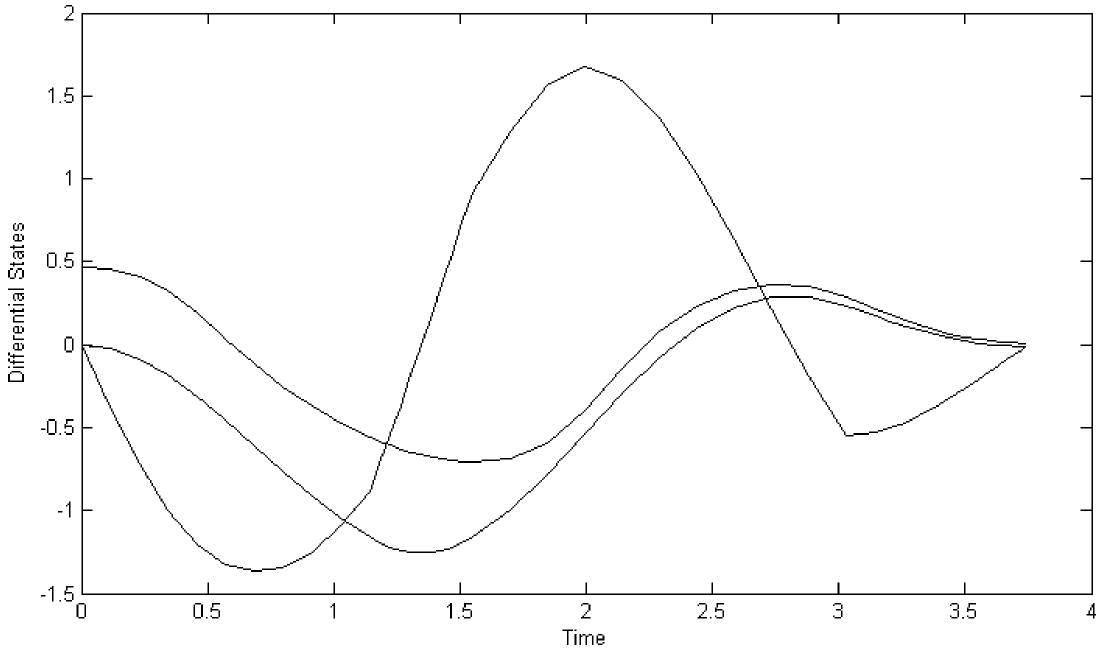

Figure 3: Differential states

To illustrate the performance of the oracle method on this application we include a Figure showing all oracle parameter updates within `MIDACO` over the optimization horizon of 8 hours. Figure 4 shows the convergence curve of all oracle parameters Ω^i (see Section 2.4) starting with the very first feasible solution which was found after about 10000 evaluations with an objective function value of about 5.87.

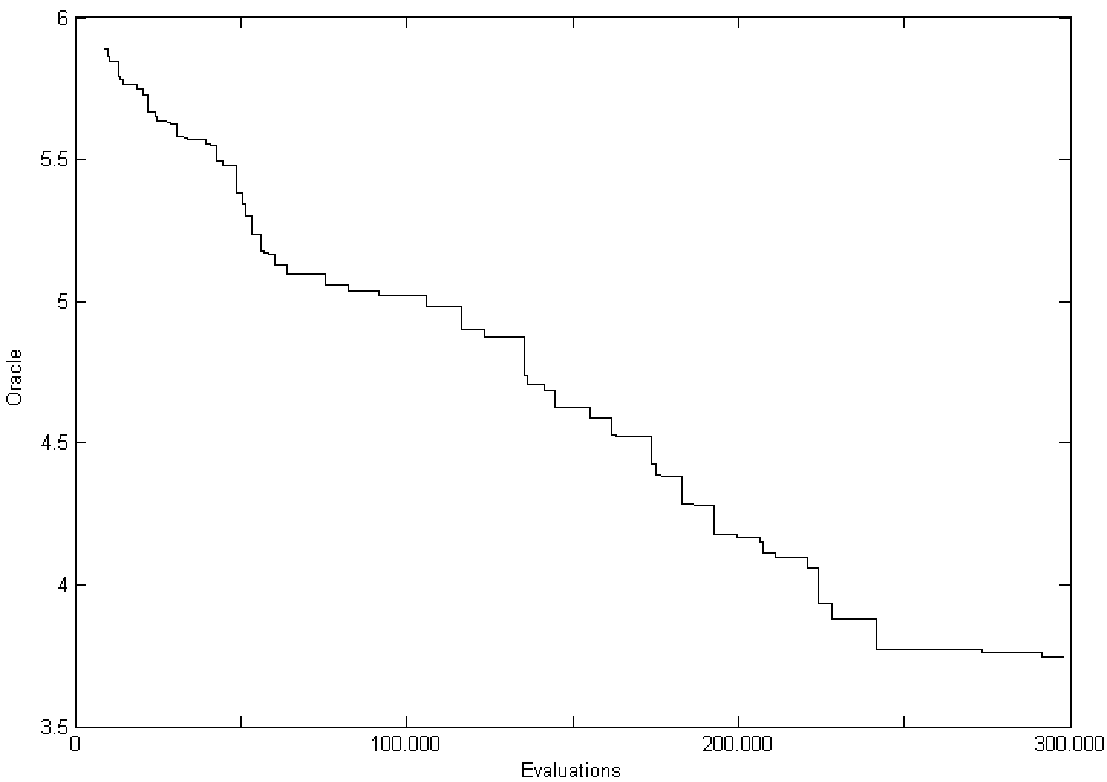

Figure 4: Oracle parameter convergence curve

From the quality of the MIDACO solution and the many oracle parameter updates shown in Figure 4 we conclude, that the oracle penalty method is capable to solve even complex problems where well known solvers fail.

4 Conclusions

A novel penalty method, intended for the use in stochastic metaheuristics, has been introduced and developed in this paper. Numerical results on a set of 60 constrained MINLP benchmark problems obtained by the oracle method have been compared to those of common penalty functions. It turned out, that the extended oracle penalty function had the highest potential in finding global optimal solutions, if the oracle parameter was selected according to a previously described update rule. Also it could be observed, that the oracle method performed very similar to a static penalty function if the oracle parameter was selected sufficiently large and was not updated.

The use of the method on three real-world applications revealed its practical strength. Either better solutions (than those reported in the literature) or the best known solutions could be obtained on all applications.

Based on those results the oracle penalty method is not only seen an alternative to the static penalty function, but also as a true alternative to concurrent advanced penalty methods. Regarding the latter the oracle method keeps the decisive advantage of only one parameter, which is easy and intuitive to handle. As for real- world applications often information about existing solution objective function values exist, the oracle penalty method seems to be highly suitable here as well.

Altogether the oracle penalty method is seen an appealing new way to handle constrained optimization problems within stochastic metaheuristics. It is easy to implement and handle, performs as robust as a static penalty function and keeps a high potential in finding global optimal solutions where other methods fail.

Acknowledgments

We would like to acknowledge the support of this research through the project *"Non-linear mixed-integer-based Optimisation Technique for Space Applications" (ESTEC/Contract No. 21943/08/NL/ST)* co-funded by ESA Networking Partnership Initiative, Astrium Limited (Stevenage, UK) and the School of Mathematics, University of Birmingham, UK.

References

[1] Asaadi, J.: A computational comparison of some non-linear programs. Math. Program. 4, 144–154 (1973)

[2] Van de Braak, G.: Das Verfahren MISQP zur gemischt ganzzahligen nichtlinearen Programmierung fuer den Entwurf elektronischer Bauteile. Dimploma Thesis, Department of Numerical and Instrumental Mathematics, University of Muenster, Germany (2001)

[3] Coello Coello, C.A.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art., Comput. Method. Appl. M. 191, 1245–1287 (2002)

[4] Coit, D.W., Smith, A.E.: Penalty guided genetic search for reliability design optimization. Comput. Ind. Eng. 30, special issue on genetic algorithms, 895–904 (1996)

[5] Dahl, H., Meeraus, A., Zenios, S.A.: Some financial optimization models: Risk managment. In Financial Optimization, S.A. Zenios, ed., Cambridge University Press, New York (1993)

[6] Dorigo, M ., Stuetzle, T.: Ant colony optimization. Cambridge, MIT Press (2004)

[7] Downs, J.J., Vogel, E.F.: Plant-wide industrial process control problem. Comput. Chem. Eng. 17, 245–255 (1993)

[8] Duran M., Grossmann, I.E.: An outer-approximation Algorithm for a class of mixed-integer nonlinear programs. Math. Program. 36, 307–339 (1986)

[9] Egea, J. A.; Rodríguez-Fernández, M.; Banga, J. R.; Martí, R. Scatter search for chemical and bio-process optimization. J. Global Optim. 37, 481–503 (2007)

[10] Exler, O., Schittkowksi, K.: A trust region SQP algorithm for mixed-integer nonlinear programming. Optimization Letters 3, 269–280 (2007)

[11] Exler, O.; Antelo, L. T.; Egea, J. A.; Alonso, A. A.; Banga, J. R. A Tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. Comput. Chem. Eng. 32, 1877–1891 (2008)

[12] Floudas, C.A., Pardalos, P.M., Adjiman, C.S., Esposito, W.R., Gumus, Z.H., Harding, S.T., Klepeis, J.L., Meyer C.A., Schweiger, C.A.: Handbook of Test Problems in Local and Global Optimization, Kluwer Academic Publishers (1999)

[13] Floudas, C.A.: Nonlinear and Mixed Integer Optimization: Fundamentals and Applications. Oxford University Press (1995)

[14] Floudas, C.A., Pardalos, P.M.: Collection of Test Problems for Constrained Global Optimization Algorithms. Lecture Notes in Computer Science 455, Springer-Verlag (1990)

[15] Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning. Kluwer Academic Publishers, Boston (1989)

[16] Glover, F., Laguna, M., Marti R.: Fundamentals of scatter search and path relinking. Control Cybern. 39, 653–684 (2000)

[17] Grossmann, I.E., Kravanja, Z.: Mixed-integer nonlinear programming: A survey of algorithms and applications. The IMA Volums in Mathematics and its Applications, Vol. 93, Large Scale Optimization with Applications. Part II: Optimal Design and Control, Biegler, Coleman eds., 73-100, Springer (1997)

[18] Gupta, O.K., Ravindran, V.: Branch and bound experiments in convex non- linear integer programming. Manage. Sci. 31, 1533–1546 (1985)

[19] Hadj-Alouane A.B., Bean, J.C.: A genetic algorithm for the multiple choice integer program. Oper. Res. 45, 92–101 (1997)

[20] Homaifar, A., Lai, S.H.Y., Qi, X.: Constrained optimization via genetic algorithms. Simulation 62, 242–254 (1994)

[21] Kaya, C.,Y., Noakes, J.,L.: A Computational Method for Time-Optimal Control. Journal of Optimization Theory and Applications 117, 69-92 (2003).

[22] Kennedy, J., Eberhart, R.: Particle swarm optimization. In proceedings of the IEEE international conference on neural networks, Piscataway, NJ, 1942–1948 (1995)

[23] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)

[24] Manne, A.S.: GAMS/MINOS: Three examples. Tech. rep., Department of Operations Research, Stanford University, (1986)

[25] Kuri Morales, A., Villegas Quezada, C.: A universal electic genetic algorithm for constraint optimization. In proceedings of the 6th european congress on intelligent techniques and soft computing, EUFIT'98, Aachen Germany, Verlag Mainz, 518–522 (1998)

[26] Michalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. In proceedings of the 4th annual conference on evolutionary programming, J.R. McDonnell, R.G. Reynolds and D.B. Fogel, eds., MIT press, Cambridge Massachusetts, 135–155, (1995)

[27] Sager, S.: mintOC, benchmark library of mixed-integer optimal control problems. (Http://mintoc.de) (2009)

[28] Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. Eur. J. Oper. Res. 185, 1155–1173 (2008)

[29] Schlüter, M., Egea, J.A., Banga, J.R.: Extended antcolony optimization for non-convex mixed integer nonlinear programming. Comput. Oper. Res. 36(7), 2217–2229 (2009)

[30] Schlüter, M., Egea, J.A., Antelo, L.T., Alonso, A.A., Banga, J.R.: An extended ant colony optimization algorithm for integrated process and control system design. Ind. Eng. Chem., accepted

[31] Schlüter, M.: MIDACO - Global Optimization Software for Mixed Integer Nonlinear Programming. (Http://www.midaco-solver.com) (2009)

[32] Smith A.E., Tate, D.M.: Genetic optimization using a penalty function. In proceedings of the 5th international conference on genetic algorithms, Stephaine Forrest, ed., San Mateo, California, Morgan Kaufmann Publishers, 499–503 (1993)

[33] Yeniay, O.: Penalty function methods for constrained optimization with genetic algorithms. Math. Comput. Appl. 10 , 45–56 (2005)

5 Appendix

A set of 60 constrained benchmark problems from the open literature has been considered to evaluate the performance of the penalty functions. Detailed information on the problems is listed in Table 15 while Table 14 contains explanations for the abbreviations used.

Table 14: Abbreviations for Table 15

Abbreviation	Explanation
Name	Problem name used in the literature
Ref	Literature reference
n	Number of variables in total
n_{int}	Number of integer variables
m_{eq}	Number of equality constraints
m	Number of constraints in total
$f(x^*)$	Best known objective function value

Table 15: Information on the benchmark problems

Name	Ref	n	n_{int}	m	m_{eq}	$f(x^*)$
asaadi 1.1	[1]	4	3	3	0	-0.409566D+02
asaadi 1.2	[1]	4	4	3	0	-0.380000D+02
asaadi 2.1	[1]	7	4	4	0	0.694903D+03
asaadi 2.2	[1]	7	7	4	0	0.700000D+03
asaadi 3.1	[1]	10	6	8	0	0.372195D+02
asaadi 3.2	[1]	10	10	8	0	0.430000D+02
van de Braak 1	[2]	7	3	2	0	0.100000D+01
van de Braak 2	[2]	7	3	4	0	-0.271828D+01
van de Braak 3	[2]	7	3	4	0	-0.898000D+02
nvs01	[18]	3	2	3	1	0.124697D+02
nvs02	[18]	8	5	3	3	0.596418D+01
nvs03	[18]	2	2	2	0	0.160000D+02
nvs05	[18]	8	2	9	4	0.547093D+01
nvs07	[18]	3	3	2	0	0.400000D+01
nvs08	[18]	3	2	3	0	0.234497D+02
nvs10	[18]	2	2	2	0	-0.310800D+03
nvs11	[18]	3	3	3	0	-0.431000D+03
nvs12	[18]	4	4	4	0	-0.481200D+03
nvs13	[18]	5	5	5	0	-0.585200D+03
nvs14	[18]	8	5	3	3	-0.403581D+00
nvs15	[18]	3	3	1	0	0.100000D+01
nvs17	[18]	7	7	7	0	-0.110040D+04
nvs18	[18]	6	6	6	0	-0.778400D+03
nvs19	[18]	8	8	8	0	-0.109840D+04
nvs20	[18]	16	5	8	0	0.230922D+03
nvs21	[18]	3	2	2	0	-0.568478D+01
nvs22	[18]	8	4	9	4	0.605822D+01
nvs23	[18]	9	9	9	0	-0.112520D+04
nvs24	[18]	10	10	10	0	-0.103320D+04
duran/grossmann 1	[8]	6	3	6	0	0.600974D+01
duran/grossmann 2	[8]	11	5	14	1	0.730357D+02
duran/grossmann 3	[8]	17	8	23	2	0.680100D+02
floudas 1	[12]	5	3	5	2	0.766718D+01
floudas 2	[12]	3	1	3	0	0.107654D+01
floudas 3	[12]	7	4	9	0	0.457958D+01
floudas 4	[12]	11	8	7	3	-0.943470D+00
floudas 5	[12]	2	2	4	0	0.310000D+02
floudas 6	[12]	2	1	3	0	-0.170000D+06
ST_E36	[10]	2	1	2	1	-0.246000D+03
ST_E38	[10]	4	2	3	0	0.719773D+04
ST_E40	[10]	4	3	5	1	0.282430D+02
ST_MIQP1	[10]	5	5	1	0	0.281000D+03
ST_MIQP2	[10]	4	4	3	0	0.200000D+01
ST_MIQP3	[10]	2	2	1	0	-0.600000D+01
ST_MIQP4	[10]	6	3	4	0	-0.457400D+04
ST_MIQP5	[10]	7	2	13	0	-0.333890D+03
ST_TEST1	[10]	5	5	1	0	0.000000D+00
ST_TEST2	[10]	6	6	2	0	-0.925000D+01
ST_TEST4	[10]	6	6	5	0	-0.700000D+01
ST_TEST5	[10]	10	10	11	0	-0.110000D+03
ST_TEST6	[10]	10	10	5	0	0.471000D+03
ST_TEST8	[10]	24	24	20	0	-0.296050D+05
ST_TESTGR1	[10]	10	10	5	0	-0.128116D+02
ST_TESTGR3	[10]	20	20	20	0	-0.205900D+02
ST_TESTPH4	[10]	3	3	10	0	-0.805000D+02
TLN2	[10]	8	8	12	0	0.230000D+01
ALAN	[24]	8	4	7	2	0.292500D+01
MEANVARX	[5]	35	14	44	8	0.143692D+02
OAER	[13]	9	3	7	3	-0.192310D+01
MIP-EX	[17]	5	3	7	0	0.350000D+01

Table 17 lists the results for all 60 problems respectively to the penalty function employed in MIDACO. According to the specific problem and penalty function employed the number of global optimal solutions and the number of feasible solutions obtained are presented. The best, worst and mean objective function value, the average amount of function evaluations and time (in seconds) is also reported over all feasible solutions. In case no feasible solution was found, this information is not available. Table 16 explains all abbreviations used in Table 17. All results were calculated on the same personal computer with 2 GHz clock rate and 2 GB RAM working memory.

Table 16: Abbreviations for Table 17

Abbreviation	Explanation
Name	Problem name used in the literature
Penalty	Penalty function used in MIDACO
Optimal	Number of global optimal solutions found out of 100 test runs
Feasible	Number of feasible solutions found out of 100 test runs
f_{best}	Best (feasible) objective function value found out of 100 test runs
f_{worst}	Worst (feasible) objective function value found out of 100 test runs
f_{mean}	Mean objective function value over all runs with a feasible solution
$\text{eval}_{\text{mean}}$	Mean number of evaluations over all runs with a feasible solution
$\text{time}_{\text{mean}}$	Mean cpu-time (in seconds) over all runs with a feasible solution
na	information is not available (in case no feasible solution was found)

Table 17: Detailed results for the constrained benchmark problems

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	$eval_{mean}$	$time_{mean}$
asaadi 1.1	$Oracle_{update}$	100	100	-40.958	-40.953	-40.955	1642	0.014
	$Oracle_{fix}$	100	100	-40.958	-40.953	-40.955	1144	0.015
	Static	100	100	-40.958	-40.953	-40.955	1125	0.013
	Death	100	100	-40.958	-40.953	-40.955	1404	0.013
	$Adaptive_1$	100	100	-40.958	-40.953	-40.955	1093	0.013
	$Adaptive_2$	100	100	-40.958	-40.953	-40.955	1051	0.013
	$Adaptive_3$	100	100	-40.958	-40.953	-40.955	1115	0.012
asaadi 1.2	$Oracle_{update}$	100	100	-38.000	-38.000	-38.000	78	0.010
	$Oracle_{fix}$	100	100	-38.000	-38.000	-38.000	72	0.010
	Static	100	100	-38.000	-38.000	-38.000	74	0.009
	Death	100	100	-38.000	-38.000	-38.000	222	0.008
	$Adaptive_1$	100	100	-38.000	-38.000	-38.000	73	0.009
	$Adaptive_2$	100	100	-38.000	-38.000	-38.000	72	0.008
	$Adaptive_3$	100	100	-38.000	-38.000	-38.000	73	0.008
asaadi 2.1	$Oracle_{update}$	100	100	694.903	694.972	694.943	8040	0.041
	$Oracle_{fix}$	100	100	694.904	694.972	694.943	1523	0.014
	Static	100	100	694.903	694.971	694.944	1804	0.014
	Death	100	100	694.903	694.972	694.941	1504	0.013
	$Adaptive_1$	100	100	694.903	694.972	694.947	1563	0.014
	$Adaptive_2$	100	100	694.904	694.972	694.947	1040	0.012
	$Adaptive_3$	100	100	694.903	694.972	694.942	1014	0.011
asaadi 2.2	$Oracle_{update}$	100	100	700.000	700.000	700.000	453	0.010
	$Oracle_{fix}$	100	100	700.000	700.000	700.000	298	0.009
	Static	100	100	700.000	700.000	700.000	264	0.009
	Death	100	100	700.000	700.000	700.000	344	0.009
	$Adaptive_1$	100	100	700.000	700.000	700.000	293	0.010
	$Adaptive_2$	100	100	700.000	700.000	700.000	286	0.009
	$Adaptive_3$	100	100	700.000	700.000	700.000	266	0.010
asaadi 3.1	$Oracle_{update}$	100	100	37.219	37.223	37.222	22882	0.122
	$Oracle_{fix}$	100	100	37.219	37.223	37.222	16919	0.101
	Static	100	100	37.219	37.223	37.222	15591	0.095
	Death	0	100	37.280	106.078	57.976	100000	0.555
	$Adaptive_1$	100	100	37.220	37.223	37.222	15003	0.118
	$Adaptive_2$	100	100	37.220	37.223	37.222	7879	0.056
	$Adaptive_3$	100	100	37.219	37.223	37.222	8700	0.059
asaadi 3.2	$Oracle_{update}$	100	100	43.000	43.000	43.000	3558	0.034
	$Oracle_{fix}$	100	100	43.000	43.000	43.000	2565	0.026
	Static	100	100	43.000	43.000	43.000	1957	0.021
	Death	85	100	43.000	87.000	47.700	29029	0.140
	$Adaptive_1$	100	100	43.000	43.000	43.000	1493	0.015
	$Adaptive_2$	100	100	43.000	43.000	43.000	1592	0.015
	$Adaptive_3$	100	100	43.000	43.000	43.000	2011	0.019
van de Braak 1	$Oracle_{update}$	100	100	1.000	1.000	1.000	10959	0.048
	$Oracle_{fix}$	100	100	1.000	1.000	1.000	12351	0.047
	Static	100	100	1.000	1.000	1.000	13943	0.052
	Death	31	100	1.000	87271.731	10520.214	55029	0.149
	$Adaptive_1$	100	100	1.000	1.000	1.000	11395	0.043
	$Adaptive_2$	100	100	1.000	1.000	1.000	8796	0.037
	$Adaptive_3$	100	100	1.000	1.000	1.000	9333	0.037
van de Braak 2	$Oracle_{update}$	100	100	-2.718	-2.718	-2.718	10125	0.040
	$Oracle_{fix}$	100	100	-2.718	-2.718	-2.718	18095	0.062
	Static	100	100	-2.718	-2.718	-2.718	3619	0.019
	Death	83	100	-2.718	99.417	5.765	29602	0.075
	$Adaptive_1$	100	100	-2.718	-2.718	-2.718	3567	0.020
	$Adaptive_2$	100	100	-2.718	-2.718	-2.718	4652	0.023
	$Adaptive_3$	100	100	-2.718	-2.718	-2.718	4349	0.021
van de Braak 3	$Oracle_{update}$	99	100	-89.800	-84.667	-89.744	19432	0.074
	$Oracle_{fix}$	98	100	-89.799	-75.817	-89.645	19143	0.067
	Static	6	100	-89.800	-75.817	-76.656	66064	0.208
	Death	4	100	-89.794	-16.535	-53.856	68646	0.172
	$Adaptive_1$	5	100	-89.795	-75.817	-76.529	66704	0.211
	$Adaptive_2$	3	100	-89.794	-75.894	-76.311	68369	0.218
	$Adaptive_3$	3	100	-89.799	-75.894	-76.311	68066	0.215

(continued)

Table 18: Detailed results for the MINLP benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
nvs01	Oracle_{update}	1	16	12.470	282.900	95.185	28134	0.062
	Oracle_{fix}	3	100	12.470	301.985	101.652	29425	0.059
	Static	4	100	12.470	874.996	106.773	29291	0.059
	Death	0	52	16.837	263.712	97.661	30000	0.050
	Adaptive_1	0	56	16.837	290.365	109.072	30000	0.051
	Adaptive_2	0	56	16.837	263.712	106.042	30000	0.051
	Adaptive_3	0	51	16.837	839.336	123.617	30000	0.051
nvs02	Oracle_{update}	0	13	5.974	6.457	6.166	80000	0.268
	Oracle_{fix}	0	100	5.998	8.008	6.633	80000	0.259
	Static	0	100	5.998	8.008	6.633	80000	0.255
	Death	0	5	6.107	11.066	8.257	80000	0.156
	Adaptive_1	0	8	6.107	12.075	8.405	80000	0.186
	Adaptive_2	0	6	6.107	11.066	7.990	80000	0.174
	Adaptive_3	0	11	6.107	11.066	7.427	80000	0.207
nvs03	Oracle_{update}	100	100	16.000	16.000	16.000	146	0.009
	Oracle_{fix}	100	100	16.000	16.000	16.000	269	0.008
	Static	100	100	16.000	16.000	16.000	255	0.008
	Death	100	100	16.000	16.000	16.000	381	0.010
	Adaptive_1	100	100	16.000	16.000	16.000	116	0.008
	Adaptive_2	100	100	16.000	16.000	16.000	228	0.008
	Adaptive_3	100	100	16.000	16.000	16.000	222	0.008
nvs05	Oracle_{update}	0	88	6.008	423.126	13.951	80000	0.309
	Oracle_{fix}	0	98	62.933	7303.811	663.450	80000	0.283
	Static	0	99	11.805	430.723	115.348	80000	0.276
	Death	0	1	40.133	40.133	40.133	80000	0.172
	Adaptive_1	0	3	101.120	237.182	171.207	80000	0.276
	Adaptive_2	0	1	176.836	176.836	176.836	80000	0.266
	Adaptive_3	0	7	18.571	393.291	112.836	80000	0.239
nvs07	Oracle_{update}	100	100	4.000	4.000	4.000	792	0.008
	Oracle_{fix}	100	100	4.000	4.000	4.000	626	0.009
	Static	100	100	4.000	4.000	4.000	708	0.009
	Death	100	100	4.000	4.000	4.000	1424	0.010
	Adaptive_1	100	100	4.000	4.000	4.000	603	0.009
	Adaptive_2	100	100	4.000	4.000	4.000	814	0.008
	Adaptive_3	100	100	4.000	4.000	4.000	811	0.011
nvs08	Oracle_{update}	100	100	23.450	23.452	23.451	3531	0.014
	Oracle_{fix}	100	100	23.450	23.452	23.451	3282	0.014
	Static	100	100	23.450	23.452	23.451	3598	0.014
	Death	58	100	23.450	25.998	23.684	19944	0.040
	Adaptive_1	100	100	23.450	23.452	23.451	3247	0.014
	Adaptive_2	100	100	23.450	23.452	23.451	2791	0.013
	Adaptive_3	100	100	23.450	23.452	23.451	3602	0.015
nvs10	Oracle_{update}	100	100	-310.800	-310.800	-310.800	143	0.008
	Oracle_{fix}	100	100	-310.800	-310.800	-310.800	138	0.008
	Static	100	100	-310.800	-310.800	-310.800	139	0.008
	Death	100	100	-310.800	-310.800	-310.800	46	0.008
	Adaptive_1	100	100	-310.800	-310.800	-310.800	76	0.008
	Adaptive_2	100	100	-310.800	-310.800	-310.800	64	0.008
	Adaptive_3	100	100	-310.800	-310.800	-310.800	97	0.008
nvs11	Oracle_{update}	100	100	-431.000	-431.000	-431.000	412	0.010
	Oracle_{fix}	100	100	-431.000	-431.000	-431.000	327	0.009
	Static	100	100	-431.000	-431.000	-431.000	325	0.009
	Death	100	100	-431.000	-431.000	-431.000	70	0.008
	Adaptive_1	100	100	-431.000	-431.000	-431.000	109	0.009
	Adaptive_2	100	100	-431.000	-431.000	-431.000	88	0.007
	Adaptive_3	100	100	-431.000	-431.000	-431.000	129	0.008
nvs12	Oracle_{update}	100	100	-481.200	-481.200	-481.200	748	0.009
	Oracle_{fix}	100	100	-481.200	-481.200	-481.200	454	0.009
	Static	100	100	-481.200	-481.200	-481.200	441	0.009
	Death	100	100	-481.200	-481.200	-481.200	95	0.008
	Adaptive_1	100	100	-481.200	-481.200	-481.200	150	0.008
	Adaptive_2	100	100	-481.200	-481.200	-481.200	134	0.008
	Adaptive_3	100	100	-481.200	-481.200	-481.200	171	0.008

(continued)

Table 19: Detailed results for the benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	$eval_{mean}$	$time_{mean}$
nvs13	$Oracle_{update}$	100	100	-585.200	-585.200	-585.200	1871	0.014
	$Oracle_{fix}$	100	100	-585.200	-585.200	-585.200	1232	0.012
	Static	100	100	-585.200	-585.200	-585.200	956	0.011
	Death	100	100	-585.200	-585.200	-585.200	425	0.009
	$Adaptive_1$	100	100	-585.200	-585.200	-585.200	434	0.010
	$Adaptive_2$	100	100	-585.200	-585.200	-585.200	481	0.009
	$Adaptive_3$	100	100	-585.200	-585.200	-585.200	471	0.010
nvs14	$Oracle_{update}$	6	98	-0.404	-0.377	-0.395	75992	0.252
	$Oracle_{fix}$	9	100	-0.404	-0.389	-0.400	76032	0.245
	Static	9	100	-0.404	-0.389	-0.400	76032	0.245
	Death	0	46	-0.401	-0.372	-0.389	80000	0.154
	$Adaptive_1$	0	49	-0.401	-0.372	-0.390	80000	0.167
	$Adaptive_2$	0	47	-0.401	-0.372	-0.389	80000	0.157
	$Adaptive_3$	4	59	-0.404	-0.372	-0.394	77235	0.188
nvs15	$Oracle_{update}$	100	100	1.000	1.000	1.000	319	0.009
	$Oracle_{fix}$	100	100	1.000	1.000	1.000	286	0.008
	Static	100	100	1.000	1.000	1.000	295	0.008
	Death	100	100	1.000	1.000	1.000	928	0.009
	$Adaptive_1$	100	100	1.000	1.000	1.000	510	0.009
	$Adaptive_2$	100	100	1.000	1.000	1.000	409	0.009
	$Adaptive_3$	100	100	1.000	1.000	1.000	236	0.009
nvs17	$Oracle_{update}$	95	100	-1100.400	-1099.000	-1100.330	16728	0.079
	$Oracle_{fix}$	100	100	-1100.400	-1100.400	-1100.400	3133	0.022
	Static	100	100	-1100.400	-1100.400	-1100.400	2453	0.019
	Death	100	100	-1100.400	-1100.400	-1100.400	2681	0.018
	$Adaptive_1$	100	100	-1100.400	-1100.400	-1100.400	1756	0.014
	$Adaptive_2$	100	100	-1100.400	-1100.400	-1100.400	2083	0.017
	$Adaptive_3$	100	100	-1100.400	-1100.400	-1100.400	1582	0.015
nvs18	$Oracle_{update}$	100	100	-778.400	-778.400	-778.400	6535	0.031
	$Oracle_{fix}$	100	100	-778.400	-778.400	-778.400	1824	0.014
	Static	100	100	-778.400	-778.400	-778.400	1422	0.013
	Death	100	100	-778.400	-778.400	-778.400	966	0.010
	$Adaptive_1$	100	100	-778.400	-778.400	-778.400	814	0.011
	$Adaptive_2$	100	100	-778.400	-778.400	-778.400	1010	0.012
	$Adaptive_3$	100	100	-778.400	-778.400	-778.400	783	0.011
nvs19	$Oracle_{update}$	99	100	-1098.400	-1097.600	-1098.392	22328	0.118
	$Oracle_{fix}$	100	100	-1098.400	-1098.400	-1098.400	4864	0.032
	Static	100	100	-1098.400	-1098.400	-1098.400	4243	0.028
	Death	100	100	-1098.400	-1098.400	-1098.400	3347	0.024
	$Adaptive_1$	100	100	-1098.400	-1098.400	-1098.400	3180	0.023
	$Adaptive_2$	100	100	-1098.400	-1098.400	-1098.400	3252	0.023
	$Adaptive_3$	100	100	-1098.400	-1098.400	-1098.400	3174	0.022
nvs20	$Oracle_{update}$	2	100	230.945	307.139	262.930	158944	1.140
	$Oracle_{fix}$	0	100	231.180	259.882	242.667	160000	1.121
	Static	3	100	230.937	259.249	240.983	157726	1.139
	Death	0	100	242.736	493.010	319.407	160000	0.944
	$Adaptive_1$	4	100	230.933	259.211	240.242	158479	1.146
	$Adaptive_2$	14	100	230.940	244.605	239.298	153730	1.123
	$Adaptive_3$	2	100	230.945	244.508	240.879	159412	1.158
nvs21	$Oracle_{update}$	88	100	-5.686	-5.265	-5.652	13667	0.034
	$Oracle_{fix}$	96	100	-5.686	-5.096	-5.665	9470	0.025
	Static	99	100	-5.686	-5.096	-5.679	8054	0.022
	Death	95	100	-5.686	-5.096	-5.661	8880	0.024
	$Adaptive_1$	98	100	-5.686	-5.096	-5.675	7328	0.021
	$Adaptive_2$	40	100	-5.686	-0.216	-5.177	19611	0.044
	$Adaptive_3$	67	100	-5.686	-4.824	-5.492	16468	0.038
nvs22	$Oracle_{update}$	89	98	6.058	139.337	7.864	30701	0.122
	$Oracle_{fix}$	0	100	8.221	597.885	181.332	80000	0.291
	Static	0	100	6.828	513.734	92.341	80000	0.279
	Death	0	7	8.367	103.916	42.941	80000	0.176
	$Adaptive_1$	0	9	8.367	97.518	31.661	80000	0.196
	$Adaptive_2$	0	8	8.367	131.718	58.004	80000	0.186
	$Adaptive_3$	0	12	10.481	135.203	49.820	80000	0.223

(continued)

Table 20: Detailed results for the benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	$eval_{mean}$	$time_{mean}$
nvs23	$Oracle_{update}$	100	100	-1125.200	-1125.200	-1125.200	14239	0.092
	$Oracle_{fix}$	100	100	-1125.200	-1125.200	-1125.200	2787	0.025
	Static	100	100	-1125.200	-1125.200	-1125.200	1918	0.019
	Death	100	100	-1125.200	-1125.200	-1125.200	4365	0.031
	$Adaptive_1$	100	100	-1125.200	-1125.200	-1125.200	1673	0.017
	$Adaptive_2$	100	100	-1125.200	-1125.200	-1125.200	1874	0.019
	$Adaptive_3$	100	100	-1125.200	-1125.200	-1125.200	1997	0.020
nvs24	$Oracle_{update}$	48	100	-1033.200	-1030.800	-1032.452	71419	0.489
	$Oracle_{fix}$	98	100	-1033.200	-1032.000	-1033.176	21248	0.152
	Static	100	100	-1033.200	-1033.200	-1033.200	16061	0.115
	Death	99	100	-1033.200	-1032.000	-1033.188	17360	0.119
	$Adaptive_1$	99	100	-1033.200	-1032.000	-1033.188	16065	0.115
	$Adaptive_2$	100	100	-1033.200	-1033.200	-1033.200	13610	0.098
	$Adaptive_3$	100	100	-1033.200	-1033.200	-1033.200	16820	0.121
duran/grossmann 1	$Oracle_{update}$	100	100	6.009	6.010	6.010	10001	0.040
	$Oracle_{fix}$	100	100	6.009	6.010	6.010	3203	0.018
	Static	100	100	6.009	6.010	6.010	4767	0.022
	Death	0	100	7.416	9.996	9.970	60000	0.167
	$Adaptive_1$	100	100	6.008	6.010	6.010	4571	0.021
	$Adaptive_2$	100	100	6.009	6.010	6.010	3628	0.019
	$Adaptive_3$	100	100	6.008	6.010	6.010	4276	0.021
duran/grossmann 2	$Oracle_{update}$	27	97	73.030	145.561	80.393	95846	0.472
	$Oracle_{fix}$	0	100	73.071	86.112	76.253	110000	0.542
	Static	1	100	73.038	86.111	78.342	109352	0.527
	Death	0	8	108.695	112.064	110.901	110000	0.299
	$Adaptive_1$	1	100	73.042	95.205	78.316	109363	0.527
	$Adaptive_2$	3	100	73.042	86.111	79.492	108519	0.523
	$Adaptive_3$	16	100	73.029	95.205	77.456	104745	0.505
duran/grossmann 3	$Oracle_{update}$	16	64	68.006	98.877	70.987	154559	1.103
	$Oracle_{fix}$	0	100	68.078	85.499	76.835	170000	1.259
	Static	0	100	69.032	99.589	79.114	170000	1.220
	Death	0	71	78.265	126.354	104.944	170000	0.690
	$Adaptive_1$	0	100	68.277	108.683	84.780	170000	1.227
	$Adaptive_2$	0	100	68.120	108.683	79.699	170000	1.228
	$Adaptive_3$	5	100	68.011	99.589	77.588	166159	1.194
floudas 1	$Oracle_{update}$	84	100	7.667	8.740	7.730	19059	0.059
	$Oracle_{fix}$	87	100	7.667	7.931	7.701	16952	0.050
	Static	78	100	7.667	7.931	7.725	21943	0.063
	Death	60	100	7.667	8.240	7.791	21837	0.045
	$Adaptive_1$	61	100	7.667	8.240	7.788	24022	0.056
	$Adaptive_2$	56	100	7.667	8.240	7.802	26920	0.064
	$Adaptive_3$	59	100	7.667	8.240	7.794	23591	0.056
floudas 2	$Oracle_{update}$	100	100	1.076	1.077	1.076	2693	0.013
	$Oracle_{fix}$	100	100	1.076	1.077	1.077	3362	0.014
	Static	100	100	1.076	1.077	1.077	2941	0.013
	Death	84	100	1.076	1.250	1.103	12826	0.027
	$Adaptive_1$	96	100	1.076	1.250	1.082	4682	0.015
	$Adaptive_2$	100	100	1.076	1.077	1.076	2414	0.013
	$Adaptive_3$	100	100	1.076	1.077	1.077	2562	0.013
floudas 3	$Oracle_{update}$	100	100	4.579	4.580	4.580	3586	0.020
	$Oracle_{fix}$	100	100	4.579	4.580	4.580	3628	0.020
	Static	100	100	4.579	4.580	4.580	2127	0.016
	Death	100	100	4.579	4.580	4.580	7932	0.031
	$Adaptive_1$	100	100	4.579	4.580	4.580	2254	0.015
	$Adaptive_2$	100	100	4.579	4.580	4.580	2123	0.014
	$Adaptive_3$	100	100	4.579	4.580	4.580	2341	0.015
floudas 4	$Oracle_{update}$	0	21	-0.875	-0.602	-0.735	110000	0.520
	$Oracle_{fix}$	0	100	-0.884	-0.627	-0.726	110000	0.501
	Static	0	100	-0.838	-0.639	-0.721	110000	0.503
	Death	0	85	-0.804	-0.602	-0.642	110000	0.283
	$Adaptive_1$	0	85	-0.804	-0.602	-0.643	110000	0.308
	$Adaptive_2$	0	85	-0.804	-0.602	-0.644	110000	0.297
	$Adaptive_3$	0	89	-0.838	-0.602	-0.661	110000	0.334

(continued)

Table 21: Detailed results for the benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	$eval_{mean}$	$time_{mean}$
floudas 5	$Oracle_{update}$	100	100	31.000	31.000	31.000	23	0.008
	$Oracle_{fix}$	100	100	31.000	31.000	31.000	37	0.008
	Static	100	100	31.000	31.000	31.000	36	0.008
	Death	100	100	31.000	31.000	31.000	79	0.008
	$Adaptive_1$	100	100	31.000	31.000	31.000	35	0.007
	$Adaptive_2$	100	100	31.000	31.000	31.000	35	0.008
	$Adaptive_3$	100	100	31.000	31.000	31.000	35	0.008
floudas 6	$Oracle_{update}$	100	100	-170000.611	-169983.032	-169993.317	916	0.010
	$Oracle_{fix}$	100	100	-170000.647	-169983.183	-169991.543	382	0.009
	Static	100	100	-170000.509	-169983.068	-169989.930	539	0.009
	Death	100	100	-170000.384	-169983.160	-169990.790	986	0.010
	$Adaptive_1$	100	100	-170000.509	-169983.068	-169990.458	471	0.009
	$Adaptive_2$	100	100	-170000.603	-169983.445	-169992.573	476	0.008
	$Adaptive_3$	100	100	-170000.452	-169983.001	-169990.942	500	0.008
ST_E36	$Oracle_{update}$	0	100	-243.857	-147.000	-184.577	20000	0.045
	$Oracle_{fix}$	0	100	-243.857	-198.795	-224.984	20000	0.043
	Static	0	100	-243.857	-198.795	-225.354	20000	0.042
	Death	0	100	-237.689	-166.508	-177.552	20000	0.040
	$Adaptive_1$	0	100	-237.689	-166.508	-178.215	20000	0.040
	$Adaptive_2$	0	99	-237.689	-166.508	-196.989	20000	0.042
	$Adaptive_3$	0	100	-237.689	-166.508	-180.756	20000	0.040
ST_E38	$Oracle_{update}$	100	100	7196.874	7198.436	7197.741	4156	0.018
	$Oracle_{fix}$	100	100	7197.081	7198.446	7198.160	10413	0.029
	Static	100	100	7197.067	7198.444	7198.138	5664	0.020
	Death	0	100	7200.070	7446.945	7347.089	40000	0.087
	$Adaptive_1$	100	100	7196.995	7198.445	7198.138	4633	0.019
	$Adaptive_2$	100	100	7197.045	7198.437	7197.946	2185	0.013
	$Adaptive_3$	100	100	7196.953	7198.445	7197.976	2601	0.013
ST_E40	$Oracle_{update}$	74	100	28.243	50.971	28.902	20837	0.053
	$Oracle_{fix}$	16	100	28.243	33.485	28.956	36303	0.080
	Static	16	100	28.243	33.485	28.952	36303	0.081
	Death	6	100	28.243	46.556	33.970	37726	0.068
	$Adaptive_1$	6	100	28.243	46.556	32.449	38028	0.072
	$Adaptive_2$	6	100	28.243	46.556	33.840	37724	0.070
	$Adaptive_3$	11	100	28.243	46.556	31.240	37250	0.075
ST_MIQP1	$Oracle_{update}$	100	100	281.000	281.000	281.000	25	0.008
	$Oracle_{fix}$	100	100	281.000	281.000	281.000	23	0.008
	Static	100	100	281.000	281.000	281.000	22	0.008
	Death	100	100	281.000	281.000	281.000	45	0.008
	$Adaptive_1$	100	100	281.000	281.000	281.000	23	0.009
	$Adaptive_2$	100	100	281.000	281.000	281.000	20	0.009
	$Adaptive_3$	100	100	281.000	281.000	281.000	22	0.007
ST_MIQP2	$Oracle_{update}$	91	92	2.000	5.000	2.033	1600	0.012
	$Oracle_{fix}$	90	92	2.000	7.000	2.087	1659	0.011
	Static	92	96	2.000	24.000	2.521	2940	0.014
	Death	18	95	2.000	7.000	4.453	36662	0.066
	$Adaptive_1$	100	100	2.000	2.000	2.000	357	0.009
	$Adaptive_2$	100	100	2.000	2.000	2.000	323	0.008
	$Adaptive_3$	100	100	2.000	2.000	2.000	431	0.009
ST_MIQP3	$Oracle_{update}$	50	100	-6.000	0.000	-3.060	11356	0.022
	$Oracle_{fix}$	52	100	-6.000	0.000	-3.240	11759	0.026
	Static	71	100	-6.000	0.000	-4.260	6660	0.017
	Death	0	100	0.000	0.000	0.000	20000	0.035
	$Adaptive_1$	72	100	-6.000	0.000	-4.320	8009	0.020
	$Adaptive_2$	69	100	-6.000	0.000	-4.140	10253	0.023
	$Adaptive_3$	71	100	-6.000	0.000	-4.260	10733	0.022
ST_MIQP4	$Oracle_{update}$	63	100	-4574.043	-4.000	-4355.223	37472	0.093
	$Oracle_{fix}$	0	100	-4573.173	-2788.504	-4302.065	60000	0.142
	Static	3	100	-4573.611	-4508.469	-4566.237	58954	0.163
	Death	0	100	0.000	0.000	0.000	60000	0.095
	$Adaptive_1$	4	100	-4574.016	-4.000	-1310.779	57954	0.133
	$Adaptive_2$	100	100	-4573.975	-4573.544	-4573.676	9351	0.032
	$Adaptive_3$	97	100	-4574.034	-4573.387	-4573.697	7507	0.027

(continued)

Table 22: Detailed results for the benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
ST_MIQP5	Oracle_{update}	18	100	-333.890	-0.026	-89.378	65417	0.222
	Oracle_{fix}	0	100	-328.184	-159.494	-253.689	70000	0.218
	Static	0	100	-320.480	-132.144	-238.474	70000	0.216
	Death	0	100	-2.947	-0.878	-1.389	70000	0.148
	Adaptive_1	0	100	-303.702	-7.087	-21.991	70000	0.211
	Adaptive_2	0	100	-325.025	-8.102	-185.534	70000	0.221
	Adaptive_3	2	100	-333.870	-161.977	-292.891	69835	0.216
ST_TEST1	Oracle_{update}	100	100	0.000	0.000	0.000	13	0.008
	Oracle_{fix}	100	100	0.000	0.000	0.000	14	0.008
	Static	100	100	0.000	0.000	0.000	13	0.008
	Death	100	100	0.000	0.000	0.000	8	0.008
	Adaptive_1	100	100	0.000	0.000	0.000	13	0.008
	Adaptive_2	100	100	0.000	0.000	0.000	13	0.007
	Adaptive_3	100	100	0.000	0.000	0.000	13	0.007
ST_TEST2	Oracle_{update}	97	97	-9.250	-9.250	-9.250	281	0.009
	Oracle_{fix}	95	95	-9.250	-9.250	-9.250	1047	0.011
	Static	100	100	-9.250	-9.250	-9.250	89	0.008
	Death	0	0	na	na	na	na	na
	Adaptive_1	100	100	-9.250	-9.250	-9.250	476	0.009
	Adaptive_2	100	100	-9.250	-9.250	-9.250	355	0.008
	Adaptive_3	100	100	-9.250	-9.250	-9.250	451	0.010
ST_TEST4	Oracle_{update}	94	100	-7.000	-5.000	-6.930	5631	0.024
	Oracle_{fix}	100	100	-7.000	-7.000	-7.000	937	0.010
	Static	100	100	-7.000	-7.000	-7.000	729	0.010
	Death	0	0	na	na	na	na	na
	Adaptive_1	100	100	-7.000	-7.000	-7.000	860	0.011
	Adaptive_2	100	100	-7.000	-7.000	-7.000	803	0.012
	Adaptive_3	100	100	-7.000	-7.000	-7.000	871	0.011
ST_TEST5	Oracle_{update}	100	100	-110.000	-110.000	-110.000	257	0.009
	Oracle_{fix}	100	100	-110.000	-110.000	-110.000	194	0.008
	Static	100	100	-110.000	-110.000	-110.000	221	0.009
	Death	100	100	-110.000	-110.000	-110.000	597	0.010
	Adaptive_1	100	100	-110.000	-110.000	-110.000	201	0.009
	Adaptive_2	100	100	-110.000	-110.000	-110.000	218	0.009
	Adaptive_3	100	100	-110.000	-110.000	-110.000	214	0.008
ST_TEST6	Oracle_{update}	100	100	471.000	471.000	471.000	367	0.010
	Oracle_{fix}	100	100	471.000	471.000	471.000	284	0.009
	Static	100	100	471.000	471.000	471.000	359	0.009
	Death	100	100	471.000	471.000	471.000	2552	0.017
	Adaptive_1	100	100	471.000	471.000	471.000	341	0.009
	Adaptive_2	100	100	471.000	471.000	471.000	275	0.010
	Adaptive_3	100	100	471.000	471.000	471.000	294	0.010
ST_TEST8	Oracle_{update}	18	94	-29605.000	20019.000	-28364.447	216275	2.147
	Oracle_{fix}	0	100	-12747.000	41715.000	9559.870	240000	2.282
	Static	0	100	-16953.000	28673.000	5143.690	240000	2.274
	Death	1	98	-29605.000	29383.000	-11421.878	239486	1.145
	Adaptive_1	3	100	-29605.000	14247.000	-22419.250	237622	1.474
	Adaptive_2	3	100	-29605.000	-7997.000	-22628.620	238601	1.413
	Adaptive_3	4	100	-29605.000	19728.000	-24619.280	235769	1.833
ST_TESTGR1	Oracle_{update}	96	100	-12.812	-12.771	-12.810	41437	0.171
	Oracle_{fix}	100	100	-12.812	-12.810	-12.811	8059	0.042
	Static	100	100	-12.812	-12.810	-12.811	7315	0.038
	Death	100	100	-12.812	-12.810	-12.811	6580	0.035
	Adaptive_1	100	100	-12.812	-12.810	-12.811	7301	0.037
	Adaptive_2	100	100	-12.812	-12.810	-12.811	4676	0.027
	Adaptive_3	100	100	-12.812	-12.810	-12.811	6209	0.033
ST_TESTGR3	Oracle_{update}	5	100	-20.590	-20.220	-20.436	196512	1.567
	Oracle_{fix}	16	100	-20.590	-20.467	-20.554	186596	1.567
	Static	73	100	-20.590	-20.570	-20.585	112338	0.905
	Death	26	100	-20.590	-20.455	-20.562	164749	1.342
	Adaptive_1	76	100	-20.590	-20.570	-20.586	110635	0.891
	Adaptive_2	97	100	-20.590	-20.570	-20.590	66634	0.544
	Adaptive_3	73	100	-20.590	-20.570	-20.585	113526	0.914

(continued)

Table 23: Detailed results for the benchmark problems (continued)

Name	Penalty	Optimal	Feasible	f_{best}	f_{worst}	f_{mean}	$eval_{mean}$	$time_{mean}$
ST_TESTPH4	$Oracle_{update}$	100	100	-80.500	-80.500	-80.500	207	0.008
	$Oracle_{fix}$	100	100	-80.500	-80.500	-80.500	213	0.008
	Static	100	100	-80.500	-80.500	-80.500	220	0.009
	Death	100	100	-80.500	-80.500	-80.500	84	0.008
	$Adaptive_1$	100	100	-80.500	-80.500	-80.500	114	0.008
	$Adaptive_2$	100	100	-80.500	-80.500	-80.500	82	0.007
	$Adaptive_3$	100	100	-80.500	-80.500	-80.500	158	0.008
TLN2	$Oracle_{update}$	100	100	2.300	2.300	2.300	614	0.010
	$Oracle_{fix}$	100	100	2.300	2.300	2.300	660	0.010
	Static	100	100	2.300	2.300	2.300	517	0.010
	Death	100	100	2.300	2.300	2.300	4767	0.020
	$Adaptive_1$	100	100	2.300	2.300	2.300	525	0.009
	$Adaptive_2$	100	100	2.300	2.300	2.300	2213	0.014
	$Adaptive_3$	100	100	2.300	2.300	2.300	476	0.011
ALAN	$Oracle_{update}$	3	36	2.924	4.212	3.032	77063	0.266
	$Oracle_{fix}$	1	100	2.925	4.218	3.419	79436	0.259
	Static	1	100	2.925	4.217	3.413	79436	0.260
	Death	0	3	2.930	2.989	2.967	80000	0.141
	$Adaptive_1$	0	6	2.930	4.208	3.229	80000	0.195
	$Adaptive_2$	0	4	2.930	2.996	2.974	80000	0.172
	$Adaptive_3$	0	10	2.930	4.219	3.246	80000	0.209
MEANVARX	$Oracle_{update}$	1	9	14.342	17.449	15.133	348623	4.194
	$Oracle_{fix}$	0	84	15.155	26.652	19.879	350000	4.595
	Static	0	87	15.152	26.153	20.002	350000	4.588
	Death	0	0	na	na	na	na	na
	$Adaptive_1$	0	0	na	na	na	na	na
	$Adaptive_2$	0	0	na	na	na	na	na
	$Adaptive_3$	0	4	16.340	24.761	21.509	350000	4.344
OAER	$Oracle_{update}$	36	91	-1.924	3.492	-0.845	70105	0.288
	$Oracle_{fix}$	1	100	-1.923	-0.001	-0.492	89797	0.340
	Static	0	100	-1.913	-0.001	-0.408	90000	0.340
	Death	1	14	-1.924	-0.001	-0.633	87904	0.250
	$Adaptive_1$	0	100	-1.595	-0.001	-0.022	90000	0.363
	$Adaptive_2$	2	100	-1.923	-0.001	-0.604	89313	0.342
	$Adaptive_3$	1	100	-1.923	-0.001	-0.526	89681	0.340
MIP-EX	$Oracle_{update}$	100	100	3.500	3.500	3.500	4324	0.021
	$Oracle_{fix}$	100	100	3.500	3.500	3.500	2366	0.013
	Static	100	100	3.500	3.500	3.500	2895	0.015
	Death	100	100	3.500	3.500	3.500	919	0.009
	$Adaptive_1$	100	100	3.500	3.500	3.500	2392	0.014
	$Adaptive_2$	100	100	3.500	3.500	3.500	1431	0.012
	$Adaptive_3$	100	100	3.500	3.500	3.500	1560	0.011