Nonlinear mixed integer based Optimization Technique for Space Applications

by

Martin Schlueter

A thesis submitted to
The University of Birmingham
for the degree of
Doctor of Philosophy

School of Mathematics
The University of Birmingham
May 2012

Abstract

In this thesis a new algorithm for mixed integer nonlinear programming (MINLP) is developed and applied to several real world applications with special focus on space applications. The algorithm is based on two main components, which are an extension of the *Ant Colony Optimization* metaheuristic and the *Oracle Penalty Method* for constraint handling. A sophisticated implementation (named MIDACO) of the algorithm is used to numerically demonstrate the usefulness and performance capabilities of the here developed novel approach on MINLP. An extensive amount of numerical results on both, comprehensive sets of benchmark problems (with up to 100 test instances) and several real world applications, are presented and compared to results obtained by concurrent methods. It can be shown, that the here developed approach is not only fully competitive with established MINLP algorithms, but is even able to outperform those regarding global optimization capabilities and cpu runtime performance. Furthermore, the algorithm is able to solve challenging space applications, that are considered here as mixed integer problems for the very first time.

Acknowledgements

Firstly I would like to thank my supervisor Dr. Jan Joachim Rückmann for enabling this thesis and his distinguished advice and continuing support. Also I would like to thank Prof. Matthias Gerdts for his support and great help on technical matters. I am very grateful to Prof. Klaus Schittkowski from the University of Bayreuth for his constant endorsement and advice over many years. Dr. Jose Alberto Egea Larrosa deserves my gratefulness for introducing me to evolutionary programming and his immense personal support. To Dr. Masaharu Munetomo and his workgroup I am much obliged for the hospitality and kindly work atmosphere, offered to me at the Hokkaido University. In alphabetical order and without any claim of completeness, I would like to thank the following colleagues and friends for many fruitful intellectual discussions: Dr. Antonio Alvarez Alonso, Marco Banse, Karl Dvorsky, Oliver Exler, Dr. Ben Fairbairn, Dr. Bjoern Huepping, Eike Huesing, Dr. Sven Joachim Kimmerle, Dr. Martin Kunkel, Gomo Panganai, Eggert Rose, Heiko Schwartz. Finally I would like to express my deepest gratitude to Tomoko Ishikawa and my Family for their support and help in uncountable ways.

I also thankfully acknowledge the financial and professional support by the European Space Agency, Astrium Limited and the University of Birmingham.

Contents

List of Tables

List of Figures

Notation

$\mathbb{R}$	set of real numbers
$\mathbb{R}_0^+$	set of positive real numbers (including zero)
$\mathbb{Z}$	set of integer numbers
$\mathbb{N}$	set of natural numbers (without zero)
$\mathbb{N}_0$	set of natural numbers (including zero)
n_{con}	dimension of continuous variables
n_{int}	dimension of integer variables
n	dimension of continuous and integer variables ($n = n_{con} + n_{int}$)
x	continuous decision variables of dimension n_{con}
y	integer decision variables of dimension n_{int}
x_{lower}	lower bounds for decision variables x
x_{upper}	upper bounds for decision variables x
y_{lower}	lower bounds for decision variables y
y_{upper}	upper bounds for decision variables y
$f(x, y)$	objective function
m_e	number of equality constraints
m	number of equality and inequality constraints ($m_e \leqslant m$)
$g(x, y)$	constraint function
$p(x, y)$	penalty function
$res(x, y)$	residuum for constraint violation
$\mathcal{P}$	continuous probability density function (cPDF)
$\mathcal{Q}$	discrete probability density function (dPDF)
$\mathcal{G}$	general probability density function (PDF)
K	feasible set

$\mathbf{G}$	generation of individual ants
$\mathbf{P}$	population of several generations
$\mathcal{S}$	solution archive of individual ants
v	size of generation $\mathbf{G}$
w	size of population $\mathbf{P}$
$\mathcal{K}$	size of solution archive $\mathcal{S}$
$\mathcal{E}$	evolutionary operator
$\mathcal{E}^{ACO}$	evolutionary operator for (mixed integer) Ant Colony Optimization
$\tilde{f}(x, y)$	fitness function
ω	weights for $\mathcal{G}$
σ	standard deviation for $\mathcal{G}$
μ	mean value for $\mathcal{G}$
$\lceil \cdot \rceil$	gaussian bracket (ceil)
$\lfloor \cdot \rfloor$	gaussian bracket (floor)

Introduction

Mathematical programming (also called optimization) is an important field in applied mathematics and is widely used in industrial and academic areas. Mixed integer nonlinear programs (MINLP) are one of the most general types of finite-dimensional, single-objective mathematical programs. Containing both, continuous and integer decision variables, and without any limitation to the complexity of either the objective function or the constraints, these problems are classified as $\mathcal{NP}$-hard (see [23]). In Definition 1 the feasible set K of a mixed integer optimization problem is introduced based on equality and inequality constraints. The feasible set K is used in the following definition of the MINLP.

Definition 1 [*Feasible Set K*]

Let g be a function of the form $g : \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}} \longrightarrow \mathbb{R} \cup \{\pm\infty\}$. Then the feasible set K is defined by m functions of type g as $K = \{(x, y) \in \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}} |\ g_i(x, y) = 0 \ \wedge\ g_j(x, y) \geqslant 0 : \ \ i = 1, ..., m_e, \ \ j = m_e + 1, ..., m\}$. Here $m_e \leqslant m$ denotes the number of equality constraints, whereas m denotes the number of equality and inequality constraints in total.

The general MINLP is formally stated in Definition 2.

Definition 2 [*Mixed Integer Nonlinear Program (MINLP)*]

Let f be a function of the form $f : \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}} \longrightarrow \mathbb{R} \cup \{\pm\infty\}$ and K be a subset of the form $K \subseteq \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}}$. Then a mixed integer nonlinear program (MINLP) is given by finding a solution $(x^*, y^*) \in K$ so that $f(x^*, y^*) \leqslant f(x, y)\ \forall (x, y) \in K$.

For more than half of a century algorithmic approaches have been developed to solve MINLP problems (see [34]). While classic approaches for MINLP (see Chapter 1) were strictly deterministic, stochastic algorithms gain more and more attention in recent years (see [36] or [54]). In this thesis, a conceptually new stochastic algorithm is developed for the general MINLP problem stated in Definition 2. It is based on a novel extension of the well known stochastic metaheuristic called *Ant Colony Optimization* (see Chapter 2) and the recently developed *Oracle Penalty Method* (see Chapter 3). As a stochastic algorithm it is fundamentally different from classic approaches on MINLP. Whereby its main advantage can be seen in its robustness regarding critical function properties (like non-convexity or discontinuities), while its main disadvantage is considered its heuristic nature. The developed algorithm is implemented in a software called MIDACO (see Chapter 4). Due to the heuristic nature of the algorithm, large parts of this thesis are dedicated to present extensive and rigorous numerical results in order to evaluate the practical usefulness and performance of the proposed algorithm. Besides numerical results on large MINLP test sets (see Chapter 5), a considerable amount of real world applications is presented in addition (see Chapter 6). In compliance with the thesis topic, the focus here on real world problems is specifically on space and aerospace applications (see Table 6.1).

The mixed integer extension on Ant Colony Optimization can be found in [43] and [44]. The Oracle Penalty Method is available as individual contribution in [45]. An introduction to the MIDACO software is given in [47] and further information can be found on the MIDACO homepage [42]. The mentioned publications do also contain large parts of the numerical results presented in Chapter 5 and Chapter 6.

The scientific relevance of the optimization algorithm developed in this thesis is also indicated by the growing number of independent academic users of the MIDACO software from various academic fields. MIDACO is for example in use at the University of Calgary (Canada), University College Dublin (Ireland), University of Illinois (USA), University of Rhode Island (USA), University of Texas (USA), Osaka University (Japan), Tokyo University of Agriculture and Technology (Japan), ESPCI Paris Tech (France), XLIM Limoges (France), TNO Delft (Netherland), University of Bayreuth (Germany), Max Planck Institut Magdeburg (Germany) and Instituto Politecnico de Setubal (Portugal).

This thesis is structured as follows: Firstly an overview on different approaches on MINLP is given in Chapter 1. Chapter 2 describes the extended *Ant Colony Optimization* algorithm, which is used for the solution of unconstrained MINLP. Chapter 3 introduces the *Oracle Penalty Method* which transforms a constrained MINLP into an unconstrained one. Combining the algorithms from Chapter 2 and Chapter 3 enables the solution of general MINLP given in Definition 2. The implementation (MIDACO) of this approach is then depicted in Chapter 4. The last two chapters of this thesis present numerical results, obtained by MIDACO, whereas Chapter 5 refers to large sets of MINLP benchmarks and Chapter 6 illustrates explicitly space and aerospace applications. Finally some conclusions are drawn and three appendices provide further details on the numerical results.

Chapter 1

Mixed Integer Nonlinear Programming

This chapter gives an overview on algorithmic approaches available for MINLP. The elementary algorithmic approaches on MINLP can be broadly classified into deterministic and stochastic ones. While some prominent deterministic approaches are historically grown and well established for MINLP, the field of stochastic algorithms for MINLP is still young and developing. The most common deterministic approaches for MINLP to find in the literature are Branch and Bound, Outer Approximation and Generalized Benders Decomposition. Here the background and fundamental concepts of those three approaches will be illustrated. Further available algorithms for MINLP, which are less prominent, are also briefly discussed. Readers with a deeper interest in MINLP algorithms are referred to Grossmann [26], who gives a very detailed introduction into deterministic MINLP algorithms. A comprehensive overview on MINLP software in general can be found in Bussieck and Vigerske [8].

1.1 Branch and Bound Method

The branch and bound (BB) method was introduced in 1960 by Land and Doig [34] for combinatorial optimization problems. Under the assumption that the discrete decision variables of the MINLP (Definition 2) are relaxable (this means, that the objective and constraint functions can also be evaluated for continuous numbers, where actually a discrete variable is expected), the BB method can also be applied to MINLP problems. By recursive branching of the original optimization problem into simplified subproblems the BB method generates a decision tree over the discrete search space of the original problem. The BB method itself only manages the branching of this tree and additional algorithms are required to solve the generated subproblems.

The nodes of the decision tree correspond to a continuous relaxation of the original combinatorial or mixed integer problem. In case of MINLP problems the relaxed NLP problem can be written as follows:

$$
\begin{aligned}
&\text{Minimize} \quad f(x,y) && (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{R}^{n_{int}} :\ n_{con},\ n_{int} \in \mathbb{N}_0),\\
&\\
&\text{subject to:} \quad g_i(x,y) \ = \ 0, && i \ = \ 1,...,m_e,\\
&\qquad\qquad\quad g_i(x,y) \ \geqslant \ 0, && i \ = \ m_e+1,...,m,\\
&\qquad\qquad\quad\ y_j \ \leqslant \ \alpha_j, && \alpha_j \in \mathbb{N},\ j \in J_{lower} \subseteq \{1,...,n_{int}\},\\
&\qquad\qquad\quad\ y_j \ \geqslant \ \beta_j, && \beta_j \in \mathbb{N},\ j \in J_{upper} \subseteq \{1,...,n_{int}\}.
\end{aligned}
\tag{1.1}
$$

In the relaxed NLP problem (1.1) the original discrete variables $y \in \mathbb{N}^{n_{int}}$ are considered as continuous variables $y \in \mathbb{R}^{n_{int}}$ and additional constraints $y_j \leqslant \alpha_j$ and $y_j \geqslant \beta_j$ branches the discrete search space, as α_j and β_j are assumed to be discrete values.

At first the BB method considers the relaxed master NLP problem without discrete branching constraints ($J_{lower} = J_{upper} = \emptyset$) which creates the root node of the decision tree. The optimal solution of the relaxed master NLP problem must then be equal or better to the optimal solution of the original MINLP problem. Therefore such a solution is considered a *lower bound* for the original problem. In contrast, any feasible discrete solution to the original MINLP problem is considered as *upper bound* which is normally obtained by some heuristic.

If the optimal solution for the relaxed master NLP is discrete or in case no feasible solution could be found, the BB method stops. In the first case the optimal solution for the MINLP is then equal to the one of the relaxed master NLP. In the second case the MINLP must be infeasible. However, these two scenarios are normally unlikely to happen.

Assuming that the optimal solution of the relaxed master NLP is continuous the BB method generates two (or more) subproblems of type (1.1) which include constraints that will exclude the previously found continuous solution. Without loss of generality let $y_j^* = c \in \mathbb{R}$ be a continuous variable within the optimal solution for the relaxed master NLP. The BB method will then branch the master problem into two subproblems, one with the constraint $y_j \leqslant \lfloor c \rfloor$ and the other with the constraint $y_j \geqslant \lceil c \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote Gaussian brackets. This procedure ensures that the optimal solution for the master problem is excluded for the subproblems and such will reveal new solutions. Based on this new solutions found for the subproblems, the lower and upper bounds can then be updated. In case a discrete optimal solution (in y) was found for a subproblem which holds a better objective function value than the current upper bound, this new solution will be updated as current best (in especially lowest) upper bound. In case the optimal solution for the subproblem is continuous and its objective function value is higher than the current lower bound, it can be updated as current best (in especially highest) lower

bound.

By further branching the subproblems in the above described manner the BB method creates a decision tree over the discrete search space of an MINLP. Comparing the solutions revealed on the subproblems with the current upper and lower bound allows then to reject some branches of this tree from further calculation efforts. Finally the subproblem will be so restricted regarding the set of constraints given by J_{lower} and J_{upper} that it will reveal a discrete solution regarding the discrete variables of the original MINLP problem.

The BB method is an exact method but will fully enumerate the discrete search space of an MINLP in the worst case. It also requires inherently that the discrete variables are relaxable. The performance of the BB method depends heavily on the MINLP problem structure and the algorithm used to solve the relaxed NLP problems. The BB method can find the global optimal solution even for non-convex MINLP problems, in case the MINLP is given in algebraic form and convex underestimators are used by the algorithm for the subproblems (see Tawarmalani and Sahinidis [50]).

1.2 Outer Approximation

The outer approximation (OA) method was first introduced for MINLP in 1986 by Duran and Grossmann [14]. For their approach the MINLP requires to be separable regarding the discrete and continuous variables. All functions need to be convex and even linear for the discrete variables. In 1994, Fletcher and Leyffer [19] were able to propose a general OA approach, which does not require separability or linear functions, however it still assumes convex functions. To illustrate the fundamental concept of the OA method we consider the special case of Duran and Grossmann here.

The OA method considers two different subproblems to the original MINLP (Definition 2). These two subproblems are the fixed NLP and the cutting plane MILP. In contrast to MINLP, MILP are considered more *easy* and can be solved for example by Branch and Bound (see Section 1.1). The fixed NLP can be stated as follows:

$$\begin{aligned} &\text{Minimize} \quad f^y(x) \qquad\qquad (x \in \mathbb{R}^{n_{con}}),\ (y \in \mathbb{Z}^{n_{int}}), \\ &\text{subject to:}\ g_i^y(x) \ = \ 0, \quad i \ = \ 1, ..., m_e, \\ &\qquad\qquad\quad g_i^y(x) \ \geqslant \ 0, \quad i \ = \ m_e + 1, ..., m. \end{aligned} \tag{1.2}$$

Note that in the fixed NLP (1.2) the discrete variables $y \in \mathbb{N}^{n_{int}}$ are fixed parameters and not variables to be optimized. Therefore $f^y(x) := f(x, y)$ and $g_i^y(x) := g_i(x, y)$ for a fixed $y \in \mathbb{N}^{n_{int}}$. The cutting plane MILP exploits the convexity of the functions $f(x, y)$ and $g(x, y)$ by replacing them with supporting hyperplanes. The cutting plane MILP can be written as follows:

$$\begin{aligned} &\text{Minimize } z \qquad (z \in \mathbb{R}),\ (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{R}^{n_{int}}), \\ &\text{subject to:} \left\{ \begin{array}{l} f(x^l, y^l) + \nabla\ f(x^l, y^l)^T \begin{pmatrix} x - x^l \\ y - y^l \end{pmatrix} \ \leqslant \ z \\ g_i(x^l, y^l) + \nabla\ g_i(x^l, y^l)^T \begin{pmatrix} x - x^l \\ y - y^l \end{pmatrix} \ = \ 0, \quad i \ = \ 1, ..., m_e \in \mathbb{N}_0 \\ g_i(x^l, y^l) + \nabla\ g_i(x^l, y^l)^T \begin{pmatrix} x - x^l \\ y - y^l \end{pmatrix} \ \geqslant \ 0, \quad i \ = \ m_e + 1, ..., m \in \mathbb{N}_0 \end{array} \right\}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad l = 1, ..., L. \end{aligned} \tag{1.3}$$

In the cutting plane MILP (1.3) the original objective function $f(x, y)$ is replaced by some real variable z. As constraints, a set of L linearizations of $f(x, y)$ and $g(x, y)$ at points $(x^l, y^l), l = 1, ..., L$ are given. Those constraints are the supporting hyperplanes referred to earlier.

The OA method works by solving the fixed NLP (1.2) and the cutting plane MILP (1.3) successively in a cycle of major iterations $l = 1, ..., L$, which generates the set of points (x^l, y^l) used for the hyperplanes in the cutting plane MILP (1.3). The OA algorithm proposed by Duran and Grossmann [14] starts by generating an initial point (x^1, y^1) as the solution of the relaxed NLP (1.1) known from the previous section. This point ($l = 1$) then generates the first cutting plane MILP (1.3) which delivers a mixed integer solution $(\tilde{x}, \tilde{y})$. Using the discrete part $\tilde{y}$ as parameters generates the fixed NLP (1.2) problem. The solution of this fixed NLP will reveal a new point (x^2, y^2) which can then be used again to augment the set of K linearization points for the cutting plane MILP and the circle continuous.

A special case may occur, when the discrete solution part $\tilde{y}$ delivered by the cutting plane MILP does not correspond to a feasible solution in the original MINLP. Hence the fixed NLP using $\tilde{y}$ as discrete parameters does not have a feasible solution. In such case an additional feasibility problem is considered which minimizes the infeasibility of the continuous variables corresponding to the fixed discrete parameters $\tilde{y}$. The gained solution is then used again to create the next cutting plane MILP.

The solutions obtained by the cutting plane MILP yield lower bounds to the original MINLP problem, while the solutions obtained by the fixed NLP yield upper bounds. The OA algorithm continuous cycling between those two subproblems, till those lower and upper bounds are within a specified tolerance. Duran and Grossmann could show in

[14] that this is guaranteed to happen in a finite number of iterations under the above assumptions for the original MINLP problem. Thus the OA method guarantees global optimality for convex MINLP problems. For non-convex MINLP problems extension of OA are known, whereas those do not guarantee global optimality.

1.3 Generalized Benders Decomposition

The generalized Benders decomposition (GBD) was introduced in 1972 by Geoffrion [24]. It is closely related to the outer approximation method but considers a different subproblem than the cutting plane MILP (1.3). Instead of using hyperplanes as in the MILP, the GBD assumes linear combinations making use of the Karush-Kuhn-Tucker condition. Furthermore it disregards the continuous variables in the subproblem, making it a discrete (or integer) linear optimization program (ILP) instead of a mixed integer linear program. This ILP problem can be formulated as follows:

$$\text{Minimize} \quad z \qquad (z \in \mathbb{R}),\ (y \in \mathbb{Z}^{n_{int}},\ x \in \mathbb{R}^{n_{con}}),$$

$$\text{subject to:} \begin{cases} f(x^l, y^l) + \nabla_y\ f(x^l, y^l)^T (y - y^l) \\ +\mu^l [\ g(x^l, y^l) + \nabla_y g(x^l, y^l)^T (y - y^l)\ ] \ \leqslant\ z, \quad l \in L_{NLP}, \\ \\ \lambda^l [\ g(x^l, y^l) + \nabla_y g(x^l, y^l)^T (y - y^l)] \ \leqslant\ 0, \qquad l \in L_{feasibility}. \end{cases} \tag{1.4}$$

In ILP (1.4) the first set of L_{NLP} constraints corresponds to the set of feasible NLP subproblems (1.2) known from the OA method. The second set of $L_{feasibility}$ constraints corresponds to a set of feasibility subproblems, which may occur when for a discrete

solution given by the ILP no feasible NLP solution exists. The feasibility subproblem aims on minimizing the NLP constraint violation and can be formulated as follows:

$$\begin{aligned} &\text{Minimize} \quad \overline{z} && (\overline{z} \in \mathbb{R}_0^+, x \in \mathbb{R}^{n_{con}}), \\ &\text{subject to:} \quad g_i^y(x) \geqslant -\overline{z}, \quad i = 1, ..., m. \end{aligned} \tag{1.5}$$

Like the OA method the GBD method solves the ILP (1.4) and the NLP subproblem (either (1.2) or (1.5)) successively in a circle. In contrast to the OA method the GBD method adds only one constraint at a time to the ILP problem, which results in a slower narrowing of the feasible search space. As a consequence the lower bound given by the ILP in the GBD method is weaker (greater or equal) to the one given by the MILP in the OA method, which may result in more major iterations of the GBD method. On the other hand the ILP problem is easier to solve than the MILP problem, because it considers only discrete variables and takes less constraints into account.

1.4 Other approaches on MINLP

Besides the above described algorithms, there exist several other, less prominent, approaches for MINLP. Those approaches are briefly described here regarding their background, algorithmic key elements and analytic properties together with relevant references.

1.4.1 Extended Cutting Plane Method

The extended cutting plane (ECP) method for MINLP was introduced by Westerlund and Petterson [52]. In contrast to the BB, OA and GBD method, it does not consider NLP

subproblems but only MILP subproblems. The ECP method iteratively solves a cutting plane MILP similar to (1.3) known from Section 1.2. Along those iterations the ECP method successively adds a linearization of the currently most violated constraint as new constraint to the MILP. This way the maximal constraint violation gets minimized and the ECP method terminates when the maximal violation lies within a specified tolerance. The ECP method is intended for convex MINLP problems only and is most suitable for problems with only a moderate degree of nonlinearity. For problems with a high degree of nonlinearity the ECP may require significant more iterations than other approaches.

1.4.2 SQP-based Method

An MINLP algorithm based on sequential quadratic programming (SQP) was proposed 2007 by Exler and Schittkowski [18]. This approach extends the concept of a trust region SQP by considering a sequence of mixed integer quadratic subproblems, instead of continuous ones. A benefit of this approach is that the discrete variables of the original MINLP problem do not need to be relaxed, as the subproblems are considered to be mixed integer. The method does not require convex functions. The method can be stabilized, if combined with BB (Section 1.1) or OA (Section 1.2). In those combined cases, the method holds a convergence proof for convex MINLP problems.

1.4.3 Mesh Adaptive Direct Search Method

A mesh adaptive direct search (MADS) algorithm for general MINLP problems was introduced by Abramson et al. [3]. Starting from a set of trial points (e.g. randomly selected starting points) the MADS algorithm applies a specified mesh on the search domain using a neighborhood function that defines the local mesh point neighbors. The MADS

algorithm then iteratively searches the mesh grid points, trying to find an improved mesh point to update its parameters. In the worst case this results in an exhaustive search of all grid points. An advantage of the MADS algorithm is that it does not require the relaxation of discrete variables. On the other hand, the user is forced to define a proper neighborhood for all discrete variables, which requires some insight in the MINLP problem. Under some smoothness assumptions for the objective and constraint functions the MADS algorithm guarantees local optimality, but not global optimality.

1.4.4 Stochastic Metaheuristics and Hybrid Algorithms

Several attempts to apply stochastic metaheuristics on MINLP have been proposed in the literature during the last decades, for example Genetic Algorithms [9] and [36] or Particle Swarm Optimization [54]. Those attempts normally extend an existing metaheuristic for purely continuous or purely combinatorial optimization problems by including one or more additional heuristics, to enable it to search the mixed integer search domain. Constraints are mostly handled via a penalty function approach (see Chapter 3) in metaheuristics. The most significant advantage of stochastic metaheuristics is their robustness regarding critical function properties (like non-convexity or discontinuities). In case of MINLP those methods hold another advantage, as they do not require the relaxation of discrete variables. On the contrary, metaheuristics do require a high amount of function evaluation due to their stochastic nature. The theoretical analysis of metaheuristics is still a young field, whereas convergence proofs are sporadically available for specific cases (e.g. proofs for evolutionary algorithms can be found in Rudolph [38]). In the case of MINLP to the knowledge of the author, no formal convergence proof for any stochastic metaheuristic is available in the literature yet.

Stochastic metaheuristics are often combined with a deterministic or local search algorithm in order to improve their local convergence behavior. Those combined algorithms are referred to as hybrid algorithm. Normally, hybrid algorithms start their optimization process by employing the general metaheuristic till some sufficient attractive solution is found. Then the hybrid algorithms starts their local search algorithm, using the previously gained sufficient attractive solution as starting point. Examples for hybrid algorithms for MINLP in the literature are OQNLP [51] and ACOmi [43]. The theoretical analysis of convergence properties of hybrid algorithms is very difficult, as those depend on the individual properties of the hybridized algorithms.

Chapter 2

Ant Colony Optimization

In this chapter the stochastic metaheuristic called *Ant Colony Optimization* (ACO) is described. This method is a nature inspired optimization algorithm which belongs to the class of evolutionary algorithms, where a population of agents share some information in order to achieve some goal. To find food, biological ants start to explore the area around their nest randomly at first. If an ant succeeds in finding a food source, it will return back to the nest, laying down a chemical pheromone trail marking its path. This pheromone trail will attract other ants to follow it in the hope of finding food again.

The basic idea of ACO algorithms is to mimic this biological behavior with artificial ants, that randomly search at first and then uses some pheromone like parameter to explore the search domain defined by an optimization problem. The ACO algorithm was first introduced in 1992 by Dorigo [12] and has attracted many attention since. While it was originally introduced for combinatorial optimization problems only, the methodology was later extended to continuous search domains, too [48]. Here a new extension to mixed integer search domains is illustrated, which was recently developed by the author in [43]

and [44]. The aim of this chapter is to give a description of the ACO algorithm applied to MINLP. A general and comprehensive introduction to ACO can be found in [13] for example.

Note that in Section 2.4 an illustrative numeric example on how ACO actually works on a simple MINLP problem can be found.

2.1 ACO for MINLP General Definitions

In order to define the ant colony algorithm for mixed integer search domains, some general definitions are required. Here those fundamental definitions are established and then used in Section 2.2 to explicitly define an ACO algorithm for MINLP. Stochastic samples are essential for the ACO algorithm, whereas those samples correspond to a given probability density function. As the case of mixed integer search domains is considered here, the definitions of two types of probability density functions are recalled; one refers to the continuous and one to the discrete domain.

Definition 2.1 [*Probability Density Function (PDF)*] *A function* $\mathcal{P} : \mathbb{R} \to \mathbb{R}_0^+$ *with*

$$\int_{-\infty}^{\infty} \mathcal{P}(t)\ dt \ = \ 1$$

is called a continuous Probability Density Function (cPDF). A function $\mathcal{Q} : \mathbb{Z} \to \mathbb{R}_0^+$ *with*

$$\sum_{d=-\infty}^{\infty} \mathcal{Q}(d) \ = \ 1$$

is called a discrete Probability Density Function (dPDF).

As any evolutionary algorithm, the ACO metaheuristic uses individual agents (called *ants*) that explore the search domain corresponding to an optimization problem. These individuals belong to different generations, which express the major iterations of the algorithm. The evolutionary aspect of the algorithm is the application of its principle law: *survival of the fittest*. The fitness of the individuals corresponds here directly to the objective function value or a penalty function. A penalty function transforms a constrained optimization problem into an unconstrained one, which is normally achieved by adding a term to the objective function that penalizes the constraint violations. Here a very broad definition of penalty function is assumed.

Definition 2.2 [*Penalty function*]
Let $(x^*, y^*) \in K$ *so that* $f(x^*, y^*) \leqslant f(x, y)\ \forall (x, y) \in K$. *A function* $p : \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}} \to \mathbb{R}$ *with* $p(x^*, y^*) \leqslant f(x, y)\ \forall (x, y) \in K$ *is called penalty function. (See Chapter 3 for examples of specific penalty functions)*

After every generation, the fittest individuals are selected and saved in some solution archive to stochastically generate the next generation of individuals. This way the algorithm aims on improving the fitness of individuals by any new generation. Next the terms individual (or *ant*), generation and fitness are defined for the mixed integer case.

Definition 2.3 [*Individual, Generation, Fitness*]
An element $(x, y) \in \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}}$ *is called individual. An individual* (x, y) *is called feasible, if* $(x, y) \in K$ *(see Definition 1). If* $(x, y) \notin K$, *the individual is called infeasible. A set* $\mathbf{G} := \{(x, y)^1, (x, y)^2, ..., (x, y)^v\}$ *is called a generation of size* $v \in \mathbb{N}$, *if all components* $x_{i=1,...,n_{con}}^{l=1,...,v}$ *are samples of a set of* n_{con} *cPDF's* $\mathcal{P}^{i=1,...,n_{con}}$ *and all components* $y_{j=1,...,n_{int}}^{l=1,...,v}$ *are samples of a set of* n_{int} *dPDF's* $\mathcal{Q}^{j=1,...,n_{int}}$ *(Note that the individuals* (x, y) *are not necessarily feasible). A function* $\tilde{f} : \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}} \to \mathbb{R}$ *is called fitness of the individual* (x, y), *if* $\tilde{f}(x, y) = -f(x, y)$ *for a MINLP (Defintion 2) with* $K = \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}}$ *and*

$\tilde{f}(x,y) = p(x,y)$ for a MINLP (Defintion 2) with $K \neq \mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}}$, where $p(x,y)$ is a penalty function (see Definition 2.2).

Next the above mentioned solution archive, which contains the fittest individuals over all generations, is defined. Further a general evolutionary operator is defined here. This operator is responsible for the creation of a generation of individuals based on the information, that is provided by the solution archive.

Definition 2.4 [*Solution Archive, Evolutionary Operator*]
A set $\mathcal{S} := \{(x,y)^1, (x,y)^2, ..., (x,y)^{\mathcal{K}}\}$ is called solution archive of size $\mathcal{K}$, if all individuals of $\mathcal{S}$ are ordered regarding their fitness, this is $\tilde{f}(x,y)^p \geqslant \tilde{f}(x,y)^q \ \forall \ p,q \in \mathbb{N} : p < q \leqslant \mathcal{K}$. A function $\mathcal{E} : (\mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}})^{\mathcal{K}} \to (\mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}})^{v}$ that creates a generation of v individuals based on $\mathcal{K}$ individuals of a solution archive $\mathcal{S}$ is called an evolutionary operator.

Note that in Definition 2.4 the size $\mathcal{K}$ of the solution archive and the size v of the generated individuals are independent parameters. However, normally it is assumed that $\mathcal{K} < v$.

2.2 An Explicit ACO Operator for MINLP

An explicit evolutionary operator (see Definition 2.4) for ant colony optimization applied to mixed integer search domains is given here. The operator, named $\mathcal{E}^{ACO}$, is based on a set of $n = n_{con} + n_{int}$ multi-kernel Gauss probability density functions $\mathcal{G}^{h=1,...,n}$. Such a multi-kernel PDF $\mathcal{G}^h$ (2.5) represents a weighted sum over $\mathcal{K}$ individual Gauss PDF's (2.6). The weights used within $\mathcal{G}^h$ are given by the parameter $\omega^h_{k=1,...,\mathcal{K}} \in \mathbb{R}$. The means for the individual Gauss PDF's within $\mathcal{G}^h$ are given by parameter $\mu^h_{k=1,...,\mathcal{K}} \in \mathbb{R}$ and the standard deviations according to every dimension $h = 1, ..., n$ are given by the parameter

$\sigma^h \in \mathbb{R}$. Then the parameter triplet $\{\omega_k^h, \mu_k^h, \sigma^h\}$ does determine the multi-kernel PDF $\mathcal{G}^h$ and therefore the algorithmic search process based on the stochastic samples given by $\mathcal{G}^h$. As those parameters therefore guide the algorithmic search process, this triplet is called *pheromone* in analogy to biological ants. The multi-kernel Gauss PDF $\mathcal{G}^h$ is given in Equation 2.5.

$$\mathcal{G}^h(t, \omega, \mu, \sigma) \;=\; \sum_{k=1}^{\mathcal{K}} \omega_k^h \; \frac{1}{\sigma^h \sqrt{2\pi}} \; e^{-\frac{(t-\mu_k^h)^2}{2\ {\sigma^h}^2}} \qquad (h = 1, ..., n_{con} + n_{int}). \tag{2.5}$$

Note that the term:

$$\frac{1}{\sigma^h \sqrt{2\pi}} \; e^{-\frac{(t-\mu_k^h)^2}{2\ {\sigma^h}^2}}, \tag{2.6}$$

within $\mathcal{G}^h$ (2.5) refers to an individual Gauss PDF. Figure 2.1 illustrates an example of a multi kernel Gauss PDF (dotted line) that is based on three individual Gauss PDF's (solid line).

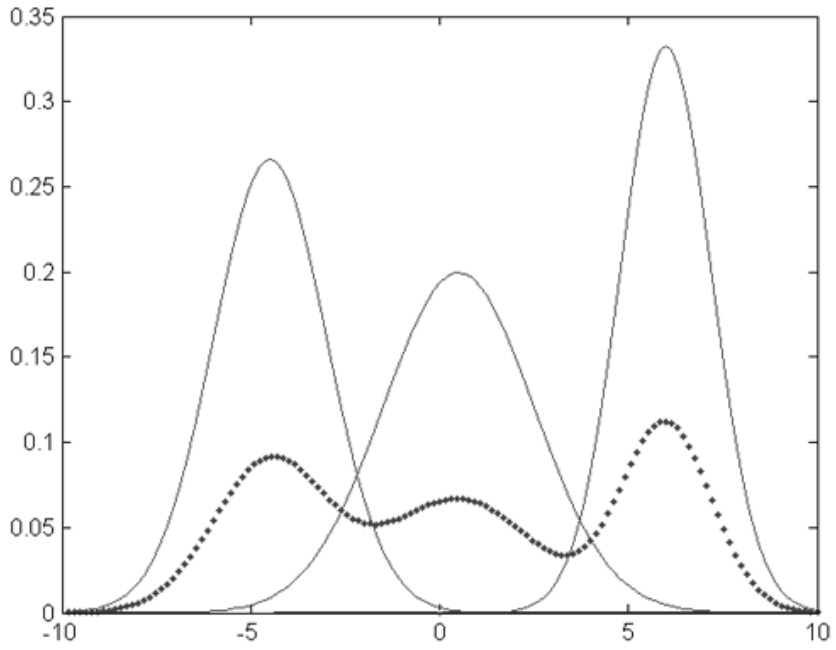

Figure 2.1: Three individual Gauss PDF's and their multi-kernel PDF

In the integer domain, a discretized version of the multi-kernel Gauss probability density function $\mathcal{G}^h(t, \omega, \mu, \sigma)$ is applied. Such discretized version accumulates the probability given by $\mathcal{G}^h(t, \omega, \mu, \sigma)$ around an integer d in the interval $[d - \frac{1}{2},\ d + \frac{1}{2}]$. The cPDF's and dPDF's described in Section 2.1 are then given by:

$$\begin{aligned} \mathcal{P}^i(t) &= \mathcal{G}^i(t, \omega, \mu, \sigma) \qquad (i = 1, ..., n_{con}), \\ \mathcal{Q}^j(d) &= \int_{d-\frac{1}{2}}^{d+\frac{1}{2}} \mathcal{G}^{n_{con}+j}(t, \omega, \mu, \sigma)\ dt \qquad (j = 1, ..., n_{int}). \end{aligned} \tag{2.7}$$

Figure 2.2 illustrates the discretized version of the multi-kernel PDF shown in Figure 2.1.

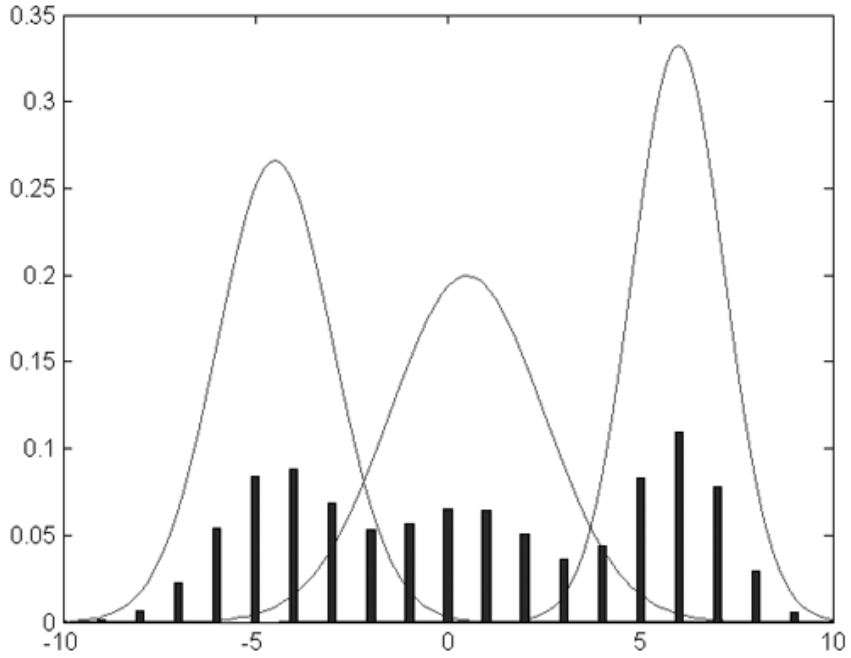

Figure 2.2: Discretized version of the multi-kernel PDF shown in Figure 2.1

The individual cPDF's $\mathcal{P}^i(t)$ and dPDF's $\mathcal{Q}^j(d)$ are determined by the following triplet of parameters: $\{\omega_k^h, \mu_k^h, \sigma^h\}$. As this triplet is therefore responsible for the direction of the search process of the ACO algorithm, which is performed by random samples based on $\mathcal{P}^i(t)$ and $\mathcal{Q}^j(d)$, it is referred to as *pheromones* in analogy to biological ants. The ω_k^h depend only on the size $\mathcal{K}$ of the solution archive $\mathcal{S}$ and act as weight within every multi-kernel Gauss PDF $\mathcal{G}^h(t, \omega, \mu, \sigma)$. Those weights are calculated by:

$$\omega_k^h = \frac{\mathcal{K} - k + 1}{\sum_{k=1}^{\mathcal{K}} k}, \tag{2.8}$$

giving the first kernel the highest probability and the last kernel the lowest probability to be selected. Note that the sum over all weights $\omega_1^h, ..., \omega_{\mathcal{K}}^h$ is assumed to be equal to one, this is $\sum_{k=1}^{\mathcal{K}} \omega_k^h = 1$

The μ_k^h represent the means of the individual Gauss PDF's within the multi-kernel PDF $\mathcal{G}^h(t, \omega, \mu, \sigma)$. They are easily calculated by the ants $(x, y)^{k=1,...,\mathcal{K}}$ stored in the solution archive $\mathcal{S}$ in respect to their dimension h as:

$$\mu_k^h = \begin{cases} x_h^k & \text{if} \quad h = 1, ..., n_{con}, \\ \\ y_{h-n_{con}}^k & \text{if} \quad h = n_{con}+1, ..., n_{con}+n_{int}. \end{cases} \tag{2.9}$$

The σ^h express the standard deviations of the individual Gauss PDF's within the multi-kernel PDF $\mathcal{G}^h(t, \omega, \mu, \sigma)$. Those are calculated based on the minimal distance $D_{\min}$ and maximal distance $D_{\max}$ between the individual dimensions h of the ants $(x, y)^{k=1,...,\mathcal{K}}$ stored in the solution archive $\mathcal{S}$. Those minimal and maximal distances are calculated by:

$$D_{\min}^h = \begin{cases} \min\{|x_h^p - x_h^q| : \ p, q \in \mathbb{N}, \ p \neq q \leqslant \mathcal{K}\} & \text{if} \quad h = 1, ..., n_{con}, \\ \\ \min\{|y_{h-n_{con}}^p - y_{h-n_{con}}^q| : \ p, q \in \mathbb{N}, \ p \neq q \leqslant \mathcal{K}\} & \text{if} \quad h = n_{con}+1, ..., n_{con}+n_{int}. \end{cases} \tag{2.10}$$

The maximal distance $D_{\max}^h$ is calculated analog as:

$$D_{\max}^h = \begin{cases} \max\{|x_h^p - x_h^q| : \ p, q \in \mathbb{N}, \ p \neq q \leqslant \mathcal{K}\} & \text{if} \quad h = 1, ..., n_{con}, \\ \\ \max\{|y_{h-n_{con}}^p - y_{h-n_{con}}^q| : \ p, q \in \mathbb{N}, \ p \neq q \leqslant \mathcal{K}\} & \text{if} \quad h = n_{con}+1, ..., n_{con}+n_{int}. \end{cases} \tag{2.11}$$

Besides the minimal and maximal distances, the standard deviations σ^h take further into account the algorithmic process, indicated by the number of generations $\#\mathbf{G}$ produced

so far. The standard deviations σ^h are then calculated by:

$$\sigma^h = \begin{cases} \frac{D^h_{\max} - D^h_{\min}}{\#\mathbf{G}} & \text{if} \quad h = 1, ..., n_{con}, \\ \\ \max\left\{ \frac{D^h_{\max} - D^h_{\min}}{\#\mathbf{G}},\ \frac{1}{\#\mathbf{G}},\ (1 - \frac{1}{\sqrt{n_{int}}})/2 \right\} & \text{if} \quad h = n_{con}+1, ..., n_{con}+n_{int}. \end{cases} \tag{2.12}$$

Note that for dimension h that correspond to the discrete search space, the standard deviation σ^h will never be smaller than $\max\{ \frac{1}{\#\mathbf{G}},\ (1-\frac{1}{\sqrt{n_{int}}})/2\}$ in case that $D^h_{\max} - D^h_{\min} = 0$ (which expresses the scenario, that all discrete variables corresponding to dimension h saved in the solution archive $\mathcal{S}$ have converged to the same integer). This is done to ensure that for discrete variables the multi-kernel PDF still samples with a sufficient standard deviation, which allows to still reach nearby integers.

Based on the above described multi-kernel PDF $\mathcal{G}^h(t, \omega, \mu, \sigma)$ (2.5) the evolutionary ACO operator $\mathcal{E}^{ACO}$ for MINLP can then explicitly be given by Definition 2.13.

Definition 2.13 [*ACO operator for MINLP*]
A function $\mathcal{E}^{ACO} : (\mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}})^{\mathcal{K}} \to (\mathbb{R}^{n_{con}} \times \mathbb{Z}^{n_{int}})^{v}$ *that creates a generation* $\mathbf{G} := \{(\tilde{x}, \tilde{y})^1, (\tilde{x}, \tilde{y})^2, ..., (\tilde{x}, \tilde{y})^v\}$ *of* v *ants, based on a set of* $\mathcal{K}$ *ants from a solution archive* $\mathcal{S} := \{(x, y)^1, (x, y)^2, ..., (x, y)^{\mathcal{K}}\}$ *is called ACO operator for MINLP, if* $\tilde{x}^{l=1,...,v}_{i=1,...,n_{con}}$ *are stochastic samples of* $\mathcal{P}^i(t)$ *and* $\tilde{y}^{l=1,...,v}_{j=1,...,n_{int}}$ *are stochastic samples of* $\mathcal{Q}^i(d)$.

Note that equation (2.5) describes only the multi-kernel Gauss PDF $\mathcal{G}^h(t, \omega, \mu, \sigma)$, that is characterized by the pheromone triplet $\{\omega^h_k, \mu^h_k, \sigma^h\}$ and to which the stochastic samples generated by the ACO algorithm correspond. However, for the numerical execution of the algorithm an additional technique is required, which actually produces stochastic samples regarding to the probability distribution given by $\mathcal{G}^h(t, \omega, \mu, \sigma)$. This may be

for example the Box-Muller method which is given in Lemma 2.14. The Box-Muller method generates a Gaussian distributed stochastic sample based on two independent and uniformly distributed random numbers.

Lemma 2.14 [*Box-Muller Method [7]*]
Let $\{u_1, u_2\} \subseteq \mathbb{R}$ *be independent random variables uniformly distributed in the left-open interval* $(0, 1]$. *Let* $\tilde{\vartheta} \in \mathbb{R}$ *be given by* $\tilde{\vartheta} := \nu + \sigma\sqrt{-2ln(u_1)}\, cos(2\pi u_2)$, *where* $\nu \in \mathbb{R}_0^+$ *and* $\sigma \in \mathbb{R}$. *Then* $\tilde{\vartheta}$ *is a random variable corresponding to a normal distribution with standard deviation* σ *and mean* ν.

A proof of Lemma 2.14 can be found in [7]. Uniformly distributed random numbers, which are required by the Box-Muller method, can be obtained by a pseudo random number generator, like for example the Xorshift [35] algorithm.

2.3 ACO for MINLP Pseudo Code

In order to illustrate the ACO for MINLP algorithm in a more compact way, a pseudo code implementation is given in Algorithm 1. The algorithm follows strictly the definitions given in Section 2.1 and Section 2.2. Note that the initial generation $\mathbf{G}^1$ of ants is sampled due to a uniform PDF, while for any later generation $\mathbf{G}^{i=2,3,4,\cdots}$ the ants are samples of the evolutionary operator $\mathcal{E}^{ACO}$ (2.13), which is the essential part of the algorithm.

Algorithm 1 *ACO for MINLP*

while stopping criteria not met **do**

(0) Set $l = 1$ and initialize a first generation $\mathbf{G}^1$ (Definition 2.3) of v individuals $(x,y)^1, (x,y)^2, ..., (x,y)^v$ based on a uniform cPDF $\mathcal{P}$ (Definition 2.1) for the continuous variables x and a uniform dPDF $\mathcal{Q}$ (Definition 2.1) for the discrete variables y.

(1) Select $\mathcal{K}$ best individuals from generation $\mathbf{G}^l$ based on their fitness value $\tilde{f}(x,y)$ (Definition 2.3) and save them as solution archive $\mathcal{S}$ (Definition 2.4).

(2) Apply evolutionary operator $\mathcal{E}^{ACO}$ (Definition 2.13) on the solution archive $\mathcal{S}$ to create the next generation $\mathbf{G}^{l+1}$ of v individuals $(x,y)^1, (x,y)^2, ..., (x,y)^v$.

(3) Introduce all individuals $(\tilde{x},\tilde{y})$ of generation $\mathbf{G}^{l+1}$ to the solution archive $\mathcal{S}$ if they have a better fitness than the lowest ranked individual in $\mathcal{S}$; this is if $\tilde{f}(\tilde{x},\tilde{y}) > \tilde{f}(x,y)^{\mathcal{K}}$. Discard those individuals from the archive, that have a worse fitness than the $\mathcal{K}$-th individual in the archive.

(4) Set $l = l + 1$ and go to (2).

end while

In Algorithm 1 the stopping criteria is not explicitly given, because this is an algorithmic choice independent of the ACO framework. Among evolutionary algorithms the two most common stopping criteria are a maximum budget of function evaluation (e.g. 1 Million function evaluation) or a maximum cpu-time budget (e.g. 1 Hour). A more sophisticated stopping criteria is the optimization progress between generations. E.g. if among a

number of generations the objective or penalty function could not be further improved.

2.4 A Illustrative Example of ACO for MINLP

An illustrative example calculation of the ACO for MINLP algorithm described in Section 2.3 is given here. This is done in order to give the reader a fast impression on how the ACO algorithm actually works on a concrete MINLP optimization problem. For simplicity reasons, a very simple MINLP is considered here. Besides the objective function it only assumes simple box constraints on the decision variables as lower and upper bound. Equation 2.15 states the mathematical formulation of the assumed example problem, where one continuous variable x_1 and one discrete variable y_1 must be optimized. Both variables must be in the range of zero to ten, where the optimal solution is zero for both variables.

$$\begin{aligned} \text{Minimize} \quad & f(x,y) \;=\; x_1 \;+\; y_1, \\ \text{subject to:} \quad & 0 \leqslant x_1 \leqslant 10, \quad \text{with } x_1 \in \mathbb{R}, \\ & 0 \leqslant y_1 \leqslant 10, \quad \text{with } y_1 \in \mathbb{Z}. \end{aligned} \tag{2.15}$$

In Subsection 2.4.1 several ACO generations are numerically demonstrated, illustrating the algorithmic procedure to solve the above example problem. As the ACO algorithm requires many function evaluation based on its stochastic nature, here only the first three and the 10-th generation $\mathbf{G}$ are demonstrated. Additionally, very small ACO parameters v and $\mathcal{K}$ are assumed, in order to enable a better overview. Here $v = 5$ and $\mathcal{K} = 3$ is assumed, which implies that in every generation $\mathbf{G}$ five individuals (also called *Ants*) are generated and the solution archive $\mathcal{S}$ contains the three best individuals found so far in the overall search process. In Subsection 2.4.1 the v individual ants of a generation $\mathbf{G}$,

the solution archive $\mathcal{S}$ and the pheromone triplet $\{\omega, \mu, \sigma\}$ are numerically displayed. As explained in the ACO pseudo code in Section 2.3, the fitness value $\tilde{f}(x, y)$ is considered rather than the objective function $f(x, y)$ in the solution archive. This is an important point for constrained MINLP, where the calculation of the fitness value is non-trivial (see Chapter 3). In this example, which only considers simple box-constraints, the fitness value is calculated as the negative of the objective function value, this is $\tilde{f}(x, y) = -f(x, y)$. Note that if a random number is sampled, which exceeds these box-constraint, the random number is refused and a new random number is sampled, as long as it is within the lower and upper bounds.

In Subsection 2.4.2 a graphical illustration of all multi-kernel Gauss PDF's $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ appearing in Subsection 2.4.1 is given.

2.4.1 Numerical Example Calculation

Based on **uniformly distributed** random samples, the very first generation $\mathbf{G}^1$ of $v = 5$ individuals is created and the $\mathcal{K} = 3$ best are stored in the solution archive $\mathcal{S}$. Based on the entries in $\mathcal{S}$ the pheromone triplet $\{\omega, \mu, \sigma\}$ (see Section 2.2) is then calculated.

Generation $\mathbf{G}^1$ of v ants

$Ant_1 = (3.4,\ 7) \quad \rightarrow \quad f(x, y) = 10.4$

$Ant_2 = (6.7,\ 9) \quad \rightarrow \quad f(x, y) = 15.7$

$Ant_3 = (1.2,\ 3) \quad \rightarrow \quad f(x, y) = 4.2$

$Ant_4 = (8.3,\ 1) \quad \rightarrow \quad f(x, y) = 9.3$

$Ant_5 = (9.5,\ 4) \quad \rightarrow \quad f(x, y) = 13.5$

Solution archive $\mathcal{S}$ of $\mathcal{K}$ ants

Place 1: $(1.2,\ 3),\ \tilde{f}(x, y) = -4.2$

Place 2: $(8.3,\ 1),\ \tilde{f}(x, y) = -9.3$

Place 3: $(3.4,\ 7),\ \tilde{f}(x, y) = -10.4$

Pheromone triplet $\{\omega, \mu, \sigma\}$ for Gauss PDF $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ based on $\mathcal{S}$

$\omega_1^1 = \frac{3}{6}$, $\omega_2^1 = \frac{2}{6}$, $\omega_3^1 = \frac{1}{6}$, $\omega_1^2 = \frac{3}{6}$, $\omega_2^2 = \frac{2}{6}$, $\omega_3^2 = \frac{1}{6}$

$\mu_1^1 = 1.2$, $\mu_2^1 = 8.3$, $\mu_3^1 = 3.4$, $\mu_1^2 = 3$, $\mu_2^2 = 1$, $\mu_3^2 = 7$

$\sigma^1 = \frac{|8.3-1.2|-|3.4-1.2|}{1} = \frac{7.1-2.2}{1} = 4.9$

$\sigma^2 = \max\{\frac{|7-1|-|3-1|}{1}, 1^{-1}, 0\} = \frac{6-2}{1} = 4$

The first generation $\mathbf{G}^1$ of ants was sampled by a uniform distribution over the full search space from zero to ten. But from the second generation $\mathbf{G}^2$ on, the ants of each generation are generated based on the evolutionary operator $\mathcal{E}^{ACO}$ (Definition 2.13). This operator applies the multi-kernel Gauss PDF's $\mathcal{G}^{1,2}$ for dimension x_1 and y_1 respectively, whereas its concrete stochastic samples depend on the pheromone triplet $\{\omega, \mu, \sigma\}$. Below the second generation $\mathbf{G}^2$ of ants is displayed. In brackets information on the concrete mean μ is given, that was used by $\mathcal{G}^{1,2}$ to generate x_1 and y_1 respectively. Note that the individuals of $\mathbf{G}^2$ are stochastic samples and thus do not exactly reproduce the mean values. For example x_1 of Ant_1 is 1.9, which is a stochastic sample of $\mathcal{G}^1$ based on the first kernel (that has probability $\omega_1^1 = \frac{3}{6}$ to be selected) using mean $\mu_1^1 = 1.2$ with a standard deviation of $\sigma^1 = 4.9$. In case of the integer variable y_1 the stochastic samples, which are continuous are discretized (by rounding them to the nearest integer). For example y_1 of Ant_4 is given as 1, which is the rounded stochastic sample (for example [1.3]) of $\mathcal{G}^2$ based on the second kernel (that has probability $\omega_2^2 = \frac{2}{6}$ to be selected) using mean $\mu_2^2 = 1$ with a standard deviation of $\sigma^1 = 4$. Note that the previous best solution (with fitness $\tilde{f}(x, y) = -4.2$) still remains in the solution archive $\mathcal{S}$, as only one better individual were created within generation $\mathbf{G}^2$.

Generation $\mathbf{G}^2$ (based on $\mathcal{E}^{ACO}$)

$Ant_1 = (1.9,\ 3) \rightarrow f(x,y) = 4.9$ (based on means: $\mu_1^1,\ \mu_1^2$)
$Ant_2 = (2.6,\ 2) \rightarrow f(x,y) = 4.6$ (based on means: $\mu_1^1,\ \mu_3^2$)
$Ant_3 = (7.5,\ 4) \rightarrow f(x,y) = 11.5$ (based on means: $\mu_2^1,\ \mu_1^2$)
$Ant_4 = (0.8,\ 1) \rightarrow f(x,y) = 1.8$ (based on means: $\mu_1^1,\ \mu_2^2$)
$Ant_5 = (4.3,\ 5) \rightarrow f(x,y) = 9.3$ (based on means: $\mu_3^1,\ \mu_1^2$)

Solution archive $\mathcal{S}$

Place 1: $(0.8,\ 1),\ \tilde{f}(x,y) = -1.8$
Place 2: $(1.2,\ 3),\ \tilde{f}(x,y) = -4.2$
Place 3: $(2.6,\ 2),\ \tilde{f}(x,y) = -4.6$

Pheromone triplet $\{\omega, \mu, \sigma\}$ for Gauss PDF $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ based on $\mathcal{S}$

$\omega_1^1 = \frac{3}{6}\ ,\quad \omega_2^1 = \frac{2}{6}\ ,\quad \omega_3^1 = \frac{1}{6}\ ,\quad \omega_1^2 = \frac{3}{6}\ ,\quad \omega_2^2 = \frac{2}{6}\ ,\quad \omega_3^2 = \frac{1}{6}$

$\mu_1^1 = 0.8\ ,\quad \mu_2^1 = 1.2\ ,\quad \mu_3^1 = 2.6\ ,\quad \mu_1^2 = 1\ ,\quad \mu_2^2 = 3\ ,\quad \mu_3^2 = 2$

$\sigma^1 = \frac{|2.6-0.8|-|1.2-0.8|}{2} = \frac{1.8-0.3}{2} = 0.75$

$\sigma^2 = \max\{\frac{|3-1|-|2-1|}{2}, 2^{-1}, 0\} = \frac{2-1}{2} = 0.5$

The third generation $\mathbf{G}^3$ of ants is created analogously to the second generation $\mathbf{G}^2$. As the pheromone triplet of $\mathbf{G}^2$ is shown above, again information on the based means can be found in brackets corresponding to each ant. Note that the previous best solution (with fitness $\tilde{f}(x,y) = -1.8$) still remains in the solution archive $\mathcal{S}$, as only two better individuals were created within generation $\mathbf{G}^3$.

Generation $\mathbf{G}^3$ (based on $\mathcal{E}^{ACO}$)

$Ant_1 = (1.9,\ 1) \rightarrow f(x,y) = 2.9$ (based on means: $\mu_2^1,\ \mu_1^2$)
$Ant_2 = (0.8,\ 4) \rightarrow f(x,y) = 4.8$ (based on means: $\mu_1^1,\ \mu_2^2$)
$Ant_3 = (1.2,\ 1) \rightarrow f(x,y) = 1.3$ (based on means: $\mu_1^1,\ \mu_1^2$)
$Ant_4 = (2.9,\ 3) \rightarrow f(x,y) = 5.9$ (based on means: $\mu_3^1,\ \mu_2^2$)
$Ant_5 = (0.6,\ 0) \rightarrow f(x,y) = 0.6$ (based on means: $\mu_1^1,\ \mu_1^2$)

Solution archive $\mathcal{S}$

Place 1: $(0.6,\ 0),\ \tilde{f}(x,y) = -0.6$
Place 2: $(1.2,\ 1),\ \tilde{f}(x,y) = -1.3$
Place 3: $(0.8,\ 1),\ \tilde{f}(x,y) = -1.8$

Pheromone triplet $\{\omega, \mu, \sigma\}$ for Gauss PDF $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ based on $\mathcal{S}$

$\omega_1^1 = \frac{3}{6}\ ,\quad \omega_2^1 = \frac{2}{6}\ ,\quad \omega_3^1 = \frac{1}{6}\ ,\quad \omega_1^2 = \frac{3}{6}\ ,\quad \omega_2^2 = \frac{2}{6}\ ,\quad \omega_3^2 = \frac{1}{6}$

$\mu_1^1 = 0.6\ ,\quad \mu_2^1 = 1.2\ ,\quad \mu_3^1 = 0.8\ ,\quad \mu_1^2 = 0\ ,\quad \mu_2^2 = 1\ ,\quad \mu_3^2 = 1$

$\sigma^1 = \frac{|1.2-0.6|-|0.8-0.6|}{3} = \frac{0.6-0.2}{3} = 0.133$

$\sigma^2 = \max\{\frac{|1-0|-|1-1|}{3}, 3^{-1}, 0\} = \frac{1-0}{3} = 0.\overline{3}$

The following generations are created completely analogously to the above scheme. Hence only one more generation is displayed here in order to illustrate, how a more advanced generation may look like. Now the tenth generation $\mathbf{G}^{10}$ is considered, were the individuals are already close to the solution of zero. Because the pheromone triplet $\{\omega, \mu, \sigma\}$ of the previous generation $\mathbf{G}^9$ is not shown here, no information on the means is attached to the ants.

Generation $\mathbf{G}^{10}$ (based on $\mathcal{E}^{ACO}$)

$Ant_1 = (0.021,\ 0) \quad \rightarrow \quad f(x,y) = 0.021$
$Ant_2 = (0.003,\ 0) \quad \rightarrow \quad f(x,y) = 0.003$
$Ant_3 = (0.017,\ 0) \quad \rightarrow \quad f(x,y) = 0.017$
$Ant_4 = (0.009,\ 1) \quad \rightarrow \quad f(x,y) = 1.009$
$Ant_5 = (0.001,\ 0) \quad \rightarrow \quad f(x,y) = 0.001$

Solution archive $\mathcal{S}$

Place 1: $(0.001,\ 0),\ \tilde{f}(x,y) = -0.001$
Place 2: $(0.003,\ 0),\ \tilde{f}(x,y) = -0.003$
Place 3: $(0.017,\ 0),\ \tilde{f}(x,y) = -0.017$

Pheromone triplet $\{\omega, \mu, \sigma\}$ for Gauss PDF $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ based on $\mathcal{S}$

$\omega_1^1 = \frac{3}{6}\ , \quad \omega_2^1 = \frac{2}{6}\ , \quad \omega_3^1 = \frac{1}{6}\ , \quad \omega_1^2 = \frac{3}{6}\ , \quad \omega_2^2 = \frac{2}{6}\ , \quad \omega_3^2 = \frac{1}{6}$

$\mu_1^1 = 0.001\ , \quad \mu_2^1 = 0.003\ , \quad \mu_3^1 = 0.017\ , \quad \mu_1^2 = 0\ , \quad \mu_2^2 = 0\ , \quad \mu_3^2 = 0$

$\sigma^1 = \frac{|0.017-0.001|-|0.003-0.001|}{10} = \frac{0.016-0.002}{10} = 0.0014$

$\sigma^2 = \max\{\frac{|0-0|-|0-0|}{10}, 10^{-1}, 0\} = 10^{-1} = 0.1$

In the pheromone triplet $\{\omega, \mu, \sigma\}$ of the tenth generation $\mathbf{G}^{10}$ a significant difference between the standard deviation $\sigma^1 = 0.0014$ for continuous variables and $\sigma^2 = 0.1$ for integer variables can be observed. This is due to the separate handling of standard deviations for continuous and integer variables described in Equation 2.12. The fixed lower bound for integer deviations is zero for this example MINLP, because $n_{int} = 1$ and therefore $(1 - \frac{1}{\sqrt{n_{int}}})/2 = 0$. However the dynamic bound $\frac{1}{\#G}$ is still active in the tenth Generation $\mathbf{G}^{10}$ were all integer solutions have already converged to zero. This moderate bound for the deviation σ^2 enables the algorithm to still have a probability to further explore the integer search space, even though all integer solutions have converged.

2.4.2 Graphical Illustration of Multi-Kernel Gauss PDF's

Here a graphical illustration of the multi-kernel Gauss PDFs's $\mathcal{G}^{1,2}(t, \omega, \mu, \sigma)$ appearing in Subsection 2.4.1 is given. Figure 2.3 displays the multi-kernel PDF $\mathcal{G}^1$ and $\mathcal{G}^2$ corresponding to the continuous domain (x_1) and the discrete domain (y_1) respectively for the 1st, 2nd, 3rd and 10th generation. Note that the multi-kernel PDF's displayed refer to the pheromone triplet $\{\omega, \mu, \sigma\}$ based on the solution archive $\mathcal{S}$ that is calculated by the ants of the corresponding generation number. In Figure 2.3 the convergence of the probability density towards the solution $(x_1^*,\ y_1^*) = (0,\ 0)$ can be well observed.

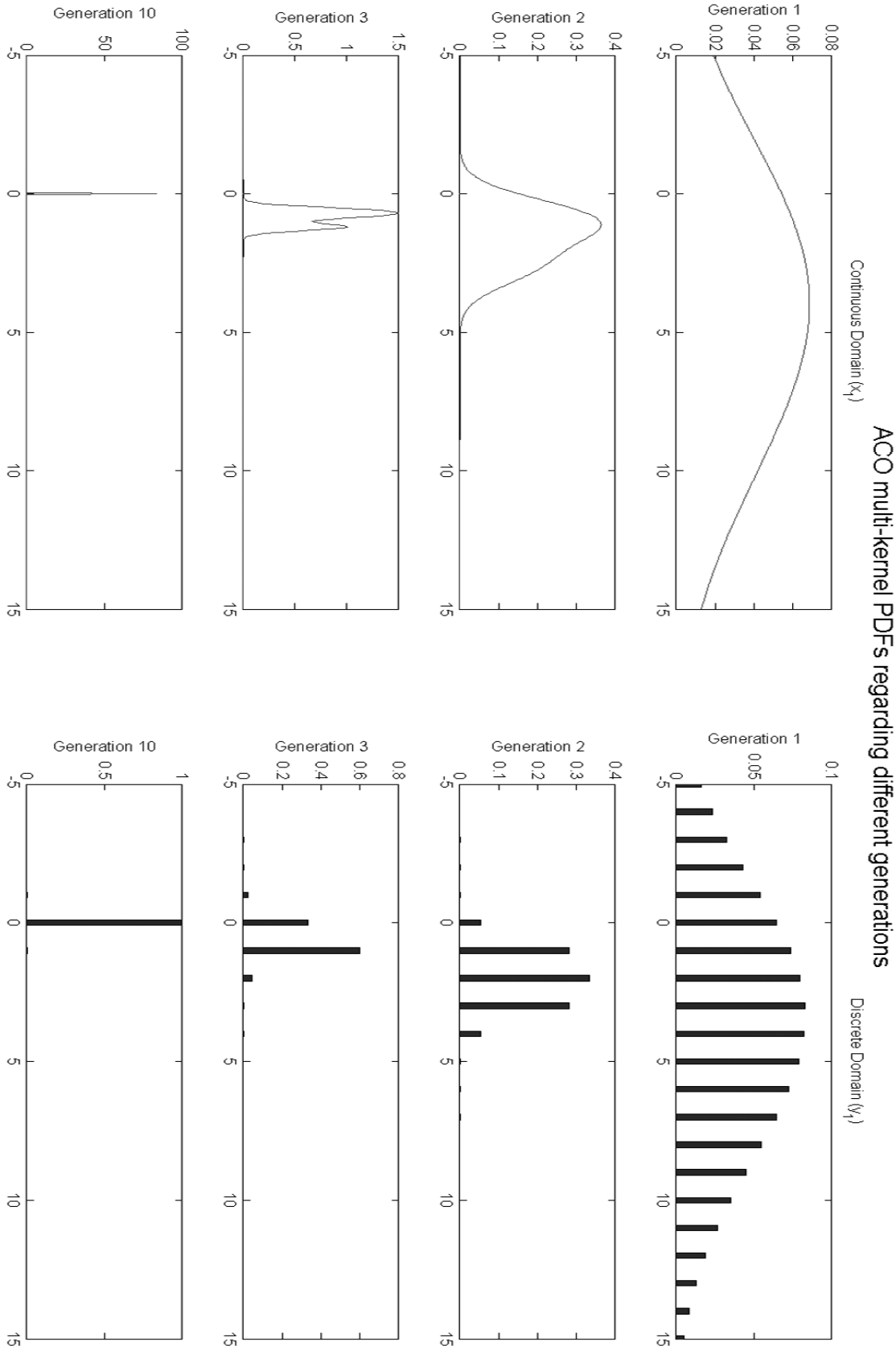

Figure 2.3: Multi-Kernel PDF's $\mathcal{G}^1$ (for x_1) and $\mathcal{G}^2$ (for y_1) from Subsection 2.4.1

Chapter 3

The Oracle Penalty Method

Penalty methods are a well known technique to handle constrained optimization problems. Those methods 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 (see [10]). The big advantage of penalty methods is their simplicity and straight forward 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 in metaheuristics is a well known problem (see [10] or [53]).

In this chapter a conceptual new penalty method, named *Oracle Penalty Method*, is presented, which has been recently developed by the author in [45]. The here proposed method is universal as it is applicable to any kind of optimization algorithm, but it is especially intended to be employed in stochastic metaheuristics. 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 a priori can therefore be seen as some kind of forecast (like an oracle in the Greek methodology). Although 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 (see numerical results in Section 5.1).

The chapter is structured as follows: Firstly, three common examples of penalty methods taken from the literature are analyzed and illustrated for the MINLP case. Secondly, the key idea of the oracle method is developed introducing a basic version of the oracle penalty method. Since the basic version lacks of robustness 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.

A numerical evaluation and comparison of the oracle penalty method versus commonly used penalty methods can be found in Section 5.1. This evaluation demonstrates, that the oracle penalty method is not only competitive with commonly used approaches, but can even outperform those.

3.1 Examples of Common Penalty Methods

Three common penalty methods (see [10]) are presented for the MINLP case and briefly analyzed. The formulations presented here follow the concept of a *residual function* $res(x, y)$ used to measure the feasibility of an iterate (x, y) to the MINLP (Definition 2). A residual function measures the constraint violations by applying a norm function over all m constraint violations of a MINLP problem (Definition 2). This approach is commonly used and some explicit residual functions based on the l^1, l^2 and l^∞ norm are listed in Table 3.1. Note that any feasible iterate (x, y) will correspond to a residual function value of zero.

Table 3.1: Examples of residual functions

Norm	residual function $res(x, y)$ for an iterate (x, y)
l^1	$res(x,y) = \sum_{i=1}^{m_e} \lvert g_i(x,y)\rvert - \sum_{i=m_e+1}^{m} \min\{0, g_i(x,y)\}$
l^2	$res(x,y) = \sqrt{\sum_{i=1}^{m_e} \lvert g_i(x,y)\rvert^2 + \sum_{i=m_e+1}^{m} \min\{0, g_i(x,y)\}^2}$
l^∞	$res(x,y) = \max\{ \lvert g_i(x,y)\rvert_{i=1,\dots,m_e}, \lvert\min\{0, g_i(x,y)\}_{i=m_e+1,\dots,m}\rvert\}$

Table 3.2 lists formulations of the ***death***, ***static*** and ***adaptive*** penalty function $p(x, y)$ for an iterate (x, y) to an MINLP problem (Defintion 2) using a residual function $res(x, y)$. The last column of Table 3.2 contains the specific parameters, required by the corresponding penalty function.

Table 3.2: Examples of common penalty functions

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

The **death** penalty function is clearly the simplest penalty function possible and it actually presents an indicator function. 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 [11].

The **static** penalty function is a sum of the objective function and the residual function

multiplied by the parameter $\mathcal{W}$. This parameter is assumed to be quite large (e.g. $\mathcal{W} = 10^9$) and enables the method to explore infeasible search regions. Even though 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. [30]. Note that the static penalty function can further be advanced, if the parameter $\mathcal{W}$ is changed/adapted after every generation (e.g. starting with a small value for $\mathcal{W}$ and increasing it over time). Here however the parameter $\mathcal{W}$ is considered to be constant and the *adaptive penalty function* (see below) is instead considered as a representant of such an adaptive method.

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 [27].

Note that the here presented penalty functions represent only the most commonly used ones within stochastic metaheuristics (see Coello Coello [10]), but other and more sophisticated approaches are known for constraint handling in optimization. Among those are for example the Filter Method (see Fletcher and Leyffer [20]) and the Augmented Lagrangian Method (see Hestenes [29]).

3.2 Development of the 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.

3.2.1 Basic Oracle Penalty Function

The key idea of the oracle penalty method is a transformation of the objective function $f(x, y)$ of the MINLP (Defintion 2) into an additional equality constraint $g_0(x, y) = f(x, y) - \Omega = 0$, where Ω is a parameter, named *oracle*. An objective function is redundant in the transformed problem and is declared as a constant zero function $\overline{f}(x, y)$. The transformed problem is then of the form:

$$\begin{aligned} \text{Minimize} \quad & \overline{f}(x,y) \;\equiv\; 0, \\ \text{subject to:} \quad & g_0(x,y) \;=\; f(x,y) - \Omega \;=\; 0, \quad \Omega \in \mathbb{R}, \\ & g_i(x,y) \;=\; 0, \quad i \;=\; 1, ..., m_e \in \mathbb{N}, \\ & g_i(x,y) \;\geqslant\; 0, \quad i \;=\; m_e + 1, ..., m \in \mathbb{N}. \end{aligned} \tag{3.1}$$

Now let (x^*, y^*) denote the global optimal solution of the MINLP (Defintion 2). Then an oracle parameter $\Omega = f(x^*, y^*)$ would directly imply, that a feasible solution of problem (3.1) is the global optimal solution of the MINLP (Defintion 2).

Assuming that for a given optimization problem the optimal objective function value $f(x^*, y^*)$ is known, the problem (3.1) holds a significant advantage compared to MINLP in Definition 2. By transforming the objective function into an equality constraint, the current progress of the algorithm in minimizing the new constraint $g_0(x, y)$ and minimizing the residual of the original constraints $g_1(x, y), ..., g_m(x, y)$ 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 (3.2) is an example of such a function:

$$p(x, y) \;=\; \mathcal{A}_\Omega(f(x, y), res(x, y)) \cdot |f(x, y) - \Omega| + (1 - \mathcal{A}_\Omega(f(x, y), res(x, y))) \cdot res(x, y), \tag{3.2}$$

where $\mathcal{A}_\Omega(f(x, y), res(x, y))$ is given by:

$$\mathcal{A}_\Omega(f(x, y), res(x, y)) \;=\; \begin{cases} 1 - \frac{1}{2\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}}} & , \text{ if } \quad res(x, y) \leqslant |f(x, y) - \Omega|, \\ \\ \frac{1}{2}\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}} & , \text{ if } \quad res(x, y) > |f(x, y) - \Omega|. \end{cases} \tag{3.3}$$

Note that the function (3.2) is actually a merit function, as it expresses a measurement for the feasibility and the distance of the current objective function value to the expected one, given as Ω. However, as the term "*penalty function*" is most widely used in stochastic metaheuristics and function (3.2) complies with Definition 2.2, it is always referred to here as penalty function. Furthermore note that the function $\mathcal{A}_\Omega(f(x, y), res(x, y))$, which assumes $f(x, y)$ and $res(x, y)$ as arguments and Ω as parameter, is substituted in the following by $\alpha := \mathcal{A}_\Omega(f(x, y), res(x, y))$ for readability reasons.

The factor α is constructed as a dynamic weight between zero and one. This factor balances the penalty function value $p(x, y)$ in respect to the relationship between $|f(x, y)-$

$\Omega|$ and $res(x, y)$. If $res(x, y) \leqslant |f(x, y) - \Omega|$ the quotient $\frac{|f(x,y)-\Omega|}{res(x,y)}$ 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 $res(x, y) > |f(x, y) - \Omega|$ the quotient $\frac{|f(x,y)-\Omega|}{res(x,y)}$ 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 3.1 illustrates the basic oracle penalty function $p(x, y)$ for an Ω parameter equal to zero according to objective function values $f(x, y) \in [-10, 10]$ and residual function values $res(x, y) \in [0, 10]$. 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.

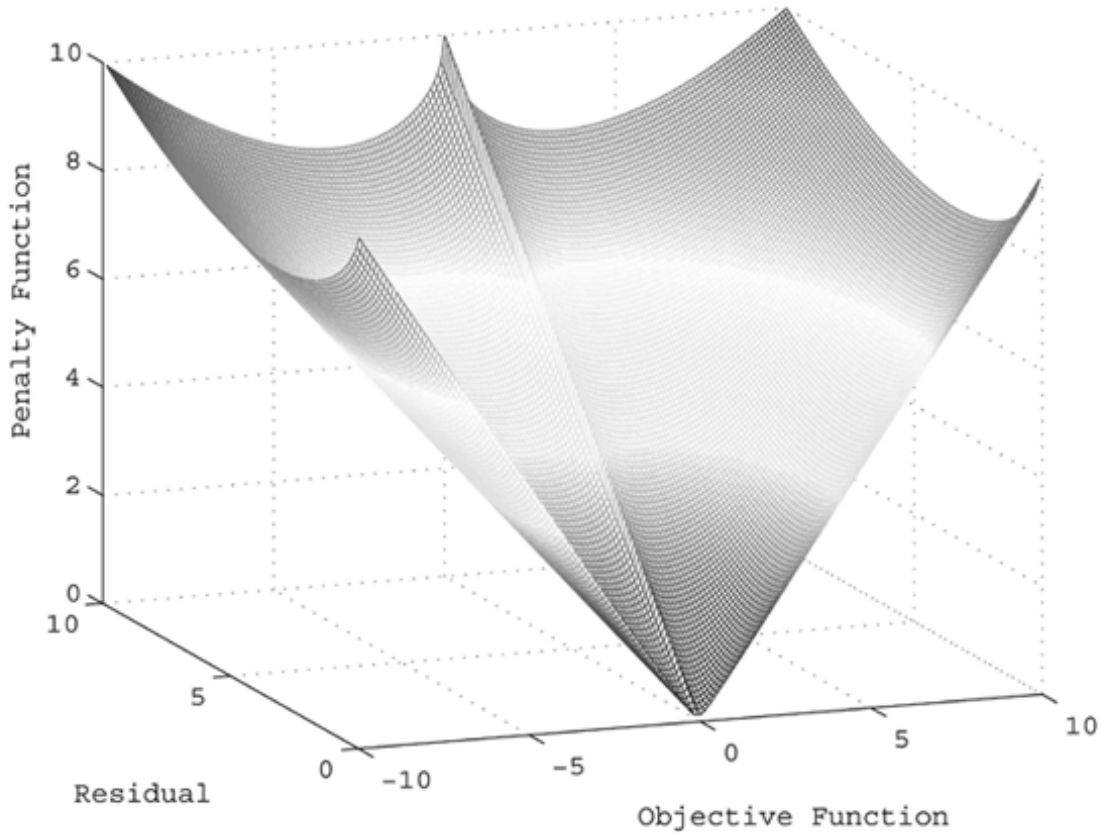

Figure 3.1: The basic oracle penalty function for $\Omega = 0$

The 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.

3.2.2 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 $f(x^*, y^*)$, this section describes modifications which make the method more robust regarding oracle parameter selection $\Omega \neq f(x^*, y^*)$. However, the modifications carried out here still assume two conditions for oracle parameters. This is $\Omega \geqslant f(x^*, y^*)$ and that at least one feasible solution $(\tilde{x}, \tilde{y})$ exists, so that $\Omega = f(\tilde{x}, \tilde{y}) \geqslant f(x^*, y^*)$. 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:

$$\mathrm{T}^{\Omega} := \{f(\tilde{x}, \tilde{y}) \mid g_i(\tilde{x}, \tilde{y}) = 0 \ (i = 1, ..., m_e) \ \wedge \ g_i(\tilde{x}, \tilde{y}) \geq 0 \ (i = m_e + 1, ..., m)\}. \quad (3.4)$$

Obviously a trust oracle can also be used as oracle parameter within the basic oracle penalty function (3.2). Such a parameter selection will guide an algorithm, minimizing the corresponding penalty function, to a feasible solution $(\tilde{x}, \tilde{y})$ with $f(\tilde{x}, \tilde{y}) = \Omega$. Nevertheless, based on the symmetric structure of the penalty function (3.2), no feasible iterate $(\overline{x}, \overline{y})$ with $f(\overline{x}, \overline{y}) < \Omega$ will be penalized lower than $(\tilde{x}, \tilde{y})$ with $f(\tilde{x}, \tilde{y}) = \Omega$.

The first modification concerns the desired property of the penalty function, to penalize any feasible iterate $(\overline{x}, \overline{y})$ with $f(\overline{x}, \overline{y}) < \Omega$ lower than a feasible iterate $(\tilde{x}, \tilde{y})$ with $f(\tilde{x}, \tilde{y}) = \Omega$ and therefore $p(\tilde{x}, \tilde{y}) = 0$. This can be easily achieved by splitting the penalty function (3.2) 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:

$$p(x,y) \;=\; \begin{cases} \alpha \cdot |f(x,y) - \Omega| + (1-\alpha) \cdot res(x,y) & \text{, if } \quad f(x,y) > \Omega \;\text{ or } \quad res(x,y) > 0, \\ -|f(x,y) - \Omega| & \text{, if } \quad f(x,y) \leqslant \Omega \;\text{ and } \; res(x,y) = 0. \end{cases} \tag{3.5}$$

Due to this modification any feasible iterate $(\overline{x}, \overline{y})$ with $f(\overline{x}, \overline{y}) < \Omega$ will be penalized with a negative value. Moreover, a smaller $f(\overline{x}, \overline{y})$ directly results in a better (lower) penalty function value, which seems quite reasonable. In other words, just the objective function $f(x, y)$ is minimized in case its value is lower or equal to Ω. However, as iterates must be compared to each other in respect to the penalty function, the term $-|f(x, y) - \Omega|$ is used here within function (3.5). Furthermore note that this first modification introduces a discontinuity in the penalty function, but as such property is only of concern to deterministic algorithms, it is not consider a major drawback here for stochastic metaheuristics (where the oracle penalty method is intended for).

The second modification concerns infeasible iterates corresponding to objective function values lower than Ω. Imagine a just slightly infeasible iterate $(\widehat{x}, \widehat{y})$ with an objective function value $f(\widehat{x}, \widehat{y})$ much lower than Ω. Such an iterate would be penalized higher than any iterate with the same residual greater than $f(\widehat{x}, \widehat{y})$ and lower than Ω. Modifying the α factor by adding a new case overcomes this undesired property. In case of an infeasible iterate $(\widehat{x}, \widehat{y})$ with $f(\widehat{x}, \widehat{y}) \leqslant \Omega$, the penalty function should penalize the iterate

with its residual $res(\widehat{x}, \widehat{y})$. This means, α is zero in such a case:

$$\alpha = \begin{cases} 1 - \frac{1}{2\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}}} & , \text{ if } \quad res(x,y) \leqslant |f(x,y)-\Omega| \quad \text{and} \quad f(x,y) > \Omega, \\ \frac{1}{2}\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}} & , \text{ if } \quad res(x,y) > |f(x,y)-\Omega| \quad \text{and} \quad f(x,y) > \Omega, \\ 0 & , \text{ if } \quad f(x,y) \leqslant \Omega. \end{cases} \tag{3.6}$$

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

This effect is caused by the construction of α and can also be observed in the shape of the basic penalty function in Figure 3.1. 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 case for α. This α case will ensure, that any iterate $(\dot{x}, \dot{y})$ with the above properties is penalized with the same value as an iterate $(\ddot{x}, \ddot{y})$ with the above properties. This means, that the mentioned area in Figure 3.1 would be plane (see Figure 3.2).

To obtain this additional case for α it is shown first that the penalty function $p(x, y)$ in (3.5), concerning only iterates (x, y) with an identical objective function value $f(x, y) > \Omega$, takes its minimum for an iterate with $res(x, y) = \frac{|f(x,y)-\Omega|}{3}$. Then the corresponding penalty function value will be calculated and used to deduce the additional case for α.

Let

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

then

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

and

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

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

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

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

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

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

The derivative of $\tilde{p}(\mathcal{R})$ with respect to $\mathcal{R}$ is given by:

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

Let

$$\frac{d}{d\mathcal{R}}\tilde{p}(\mathcal{R}) = 0,$$

then

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

$$\iff \quad \frac{|f(x,y)-\Omega|}{\sqrt{\mathcal{R}}} = 3\sqrt{\mathcal{R}}$$

$$\iff \quad \mathcal{R} = \frac{|f(x,y)-\Omega|}{3}.$$

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

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

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

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

then

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

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

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

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

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

Now the penalty function (3.5) (under the above assumptions) is set equal to the above optimal penalty function value to deduce α. Let

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

then

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

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

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

This α factor can now be applied as additional case within (3.6) for iterates (x, y) with $res(x, y) \leqslant |f(x, y) - \Omega|$ and $f(x, y) > \Omega$. All iterates (x, y) with identical objective function value $f(x, y) > \Omega$ and $res(x, y) \leqslant \frac{|f(x,y)-\Omega|}{3}$ will then be penalized with $|f(x, y) - \Omega| \cdot \frac{6\sqrt{3}-2}{6\sqrt{3}}$.

3.2.3 Extended Oracle Penalty Function

In this section the basic oracle penalty function, presented in Subsection 3.2.1, is extended by the three modification explained in Subsection 3.2.2. The extended oracle penalty function is of the form:

$$p(x,y) \;=\; \begin{cases} \alpha \cdot |f(x,y) - \Omega| + (1-\alpha) \cdot res(x,y) & , \text{ if } \quad f(x,y) > \Omega \;\text{ or } \quad res(x,y) > 0, \\ -|f(x,y) - \Omega| & , \text{ if } \quad f(x,y) \leqslant \Omega \;\text{ and } \; res(x,y) = 0. \end{cases} \tag{3.7}$$

where α is given by:

$$\alpha \;=\; \begin{cases} \frac{|f(x,y)-\Omega| \cdot \frac{6\sqrt{3}-2}{6\sqrt{3}} - res(x,y)}{|f(x,y)-\Omega| - res(x,y)} & , \text{ if } \quad f(x,y) > \Omega \quad \text{and} \quad res(x,y) < \frac{|f(x,y)-\Omega|}{3}, \\ 1 - \frac{1}{2\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}}} & , \text{ if } \quad f(x,y) > \Omega \quad \text{and} \quad \frac{|f(x,y)-\Omega|}{3} \leqslant res(x,y) \leqslant |f(x,y) - \Omega|, \\ \frac{1}{2}\sqrt{\frac{|f(x,y)-\Omega|}{res(x,y)}} & , \text{ if } \quad f(x,y) > \Omega \quad \text{and} \quad res(x,y) > |f(x,y) - \Omega|, \\ 0 & , \text{ if } \quad f(x,y) \leqslant \Omega. \end{cases} \tag{3.8}$$

Figure 3.2 illustrates the extended oracle penalty function $p(x, y)$ for a parameter Ω equal to zero according to objective function values $f(x, y) \in [-10, 10]$ and residual function values $res(x, y) \in [0, 10]$. Again, 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 ($\Omega < 0$).

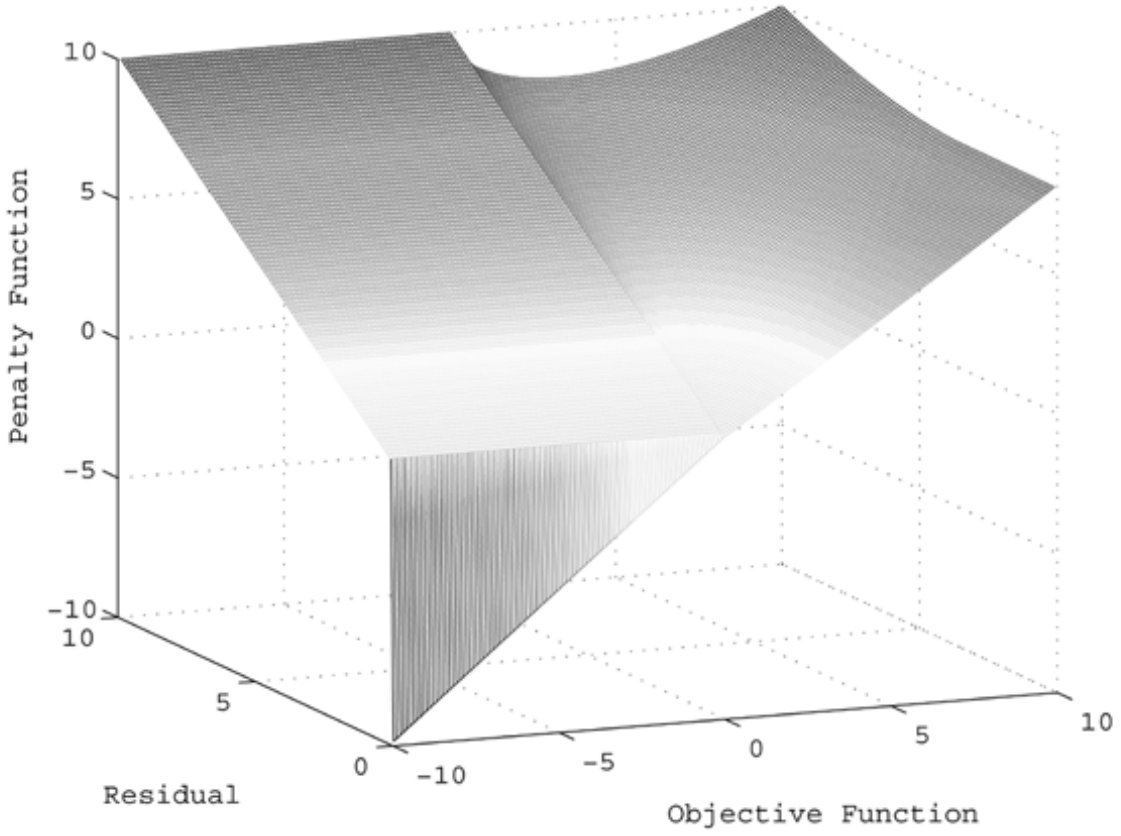

Figure 3.2: The extended oracle penalty function for $\Omega = 0$

Now it is explained how the three modifications can be observed in the shape of the extended oracle penalty function, illustrated in Figure 3.2. The first modification, concerning feasible iterates $(\overline{x}, \overline{y})$ with $f(\overline{x}, \overline{y}) < \Omega$, corresponds to the vertical triangular (in the left front), penalizing those iterates with a negative penalty function value. The second modification, concerning infeasible iterates $(\widehat{x}, \widehat{y})$ with $f(\widehat{x}, \widehat{y}) \leqslant \Omega$, corresponds to the plane area (in the left back), penalizing those iterates with their residual function value. The third modification, concerning iterates $(\dot{x}, \dot{y})$ with $f(\dot{x}, \dot{y}) > \Omega$ and

$res(\dot{x}, \dot{y}) \leqslant \frac{|f(\dot{x},\dot{y})-\Omega|}{3}$, 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 $(\ddot{x}, \ddot{y})$ with $f(\ddot{x}, \ddot{y}) = f(\dot{x}, \dot{y})$ and $res(\ddot{x}, \ddot{y}) = \frac{|f(\ddot{x},\ddot{y})-\Omega|}{3}$.

As explained in Subsection 3.2.4 those modifications are intended to make the method robust regarding trust oracles (3.4). 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 $\Omega > f(x, y)$ for any iterate (x, y). 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 $res(x, y) \geqslant 0$, while feasible iterates (x, y) will be penalized with the negative distance $-|f(x, y) - \Omega|$. Hence, for sufficiently large oracle parameters the extended oracle penalty function will act very similar to a static penalty function (see Table 3.2).

3.2.4 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! Here an optimization run is defined as a set of successive performed generations of the algorithm (see Definition 2.3).

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 $\Omega > f(x, y)$ for any iterate (x, y) to a given problem. As explained in Subsection 3.2.3 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. 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 $\Omega^{i=2,3,4,\cdots}$ used for any further optimization run should then be calculated by the following update rule:

$$\Omega^i = \begin{cases} f^{i-1} & \text{,if } \; f^{i-1} < \Omega^{i-1} \quad \text{and} \quad res^{i-1} = 0, \\ \Omega^{i-1} & \text{,else.} \end{cases} \tag{3.9}$$

According to (3.9) 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.

Note that the update rule presented in equation (3.9) essentially requires to stay feasible during the algorithmic restarts, which can be slow. Other update rules, which use more ambitious oracle parameters, which do not necessarily belong to the set of trust oracles T^{Ω} (3.4), might be possible, but are not recommended here, as they might be cause the

algorithm to get stuck in finding an impossible solution.

For a specific problem an intuitive initialization of the very first oracle parameter Ω^1 by the user is possible as well. 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 2 gives a pseudo-code implementation of the extended oracle penalty function. For a given objective function value $f(x, y)$, a residual value $res(x, y)$, an oracle Ω and some tolerance $acc \geqslant 0$ (where acc is the user defined tolerance for the constraint violation), this algorithm calculates the corresponding penalty function value $p(x, y)$. Due to the `if`-clauses in this implementation, the computational expensive α parameters are only calculated case dependent and if necessary.

Algorithm 2 `Extended oracle penalty function`

if $f(x,y) \leqslant \Omega$ **and** $res(x,y) \leqslant acc$ **then**

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

else

if $f(x,y) \leqslant \Omega$ **then**

$p(x,y) = res(x,y)$

else

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

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

end if

if $res(x,y) \geqslant \frac{f(x,y)-\Omega}{3}$ **and** $res(x,y) \leqslant (f(x,y)-\Omega)$ **then**

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

end if

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

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

end if

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

end if

end if

return $p(x,y)$

Implementations of the extended oracle penalty method in the programming languages `Fortran`, `C/C++` and `Matlab` can be found online [42] and can be downloaded.

Chapter 4

MIDACO Software

This section provides information on the software named MIDACO, which stands for *Mixed Integer Distributed Ant Colony Optimization*. MIDACO implements the extended ACO algorithm for MINLP introduced in Chapter 2 (see also [43] and [44]) in combination with the oracle penalty method described in Chapter 3 (see also [45]). For practical efficiency reasons, the software is further enhanced by some ACO relevant heuristics, which can be found in [44]. MIDACO solves the general MINLP (Definition 2) described in the Introduction. In addition to the feasible set K (Definition 1), MIDACO assumes a set of lower and upper bounds ($\{x_{lower}, y_{lower}\}$ and $\{x_{upper}, y_{upper}\}$) for the decision variables x and y. This additional set of constraints is also known as box constraints. Equation (4.1) illustrates the MINLP as assumed by MIDACO. Note that functions $f(x, y)$ and $g(x, y)$ are considered as *black box* functions by MIDACO. This means that the actual function calculation is considered to happen in a virtual black box (for example some software library) without any inside knowledge and only the function values are returned, after the actual calculation has happened.

Based on this black box concept, MIDACO does not require any assumptions on the objective or constraint functions (like convexity, differentiability or continuity) at all and can therefore be applied on a wide range of problems.

$$\begin{aligned}
&\text{Minimize} \quad f(x,y) \qquad\qquad (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{Z}^{n_{int}} :\ n_{con},\ n_{int} \in \mathbb{N}_0),\\
&\\
&\text{subject to:}\ \ g_i(x,y) \ = \ 0, \quad (i \ = \ 1,...,m_e \in \mathbb{N}_0),\\
&\qquad\qquad\ \ g_j(x,y) \ \geqslant \ 0, \quad (j \ = \ m_e+1,...,m \in \mathbb{N}_0),\\
&\\
&\text{and box constraints:}\ \ x_{lower} \leqslant x \leqslant x_{upper}, \quad (x_{lower}, x_{upper} \in \mathbb{R}^{n_{con}} : x_{lower} \leqslant x_{upper}),\\
&\qquad\qquad\qquad\qquad\ \ y_{lower} \leqslant y \leqslant y_{upper}, \quad (y_{lower}, y_{upper} \in \mathbb{Z}^{n_{int}} : y_{lower} \leqslant y_{upper}).
\end{aligned} \tag{4.1}$$

MIDACO has been developed for over five years now and is originally written in Fortran77. It has furthermore a C translation and a Matlab and MS-Excel gateway. The software is entirely written from scratch and does not require any dependencies, like libraries, external routines or compiler dependent random number generators. Uniform random numbers are generated by an internal implementation of a Xorshift generator [35] and transformed to normal distributed ones via the Box-Muller [7] method (see Lemma 2.14). MIDACO does not relax discrete optimization variables, this means objective and constraint functions are evaluated only at integer points for discrete decision variables. A main feature of the software is its user friendliness and easy compilation, since it is distributed within a single file for Fortran (`midaco.f`) and C (`midaco.c`) and within only two separated files for Matlab (`midaco.m` + `midacox.c`). The software has been successfully tested with different compilers (g77, gFortran, g95, gcc, ifort, NAG-Fortran) and on different platforms (Windows, Linux, Mac).

The software handles all its parameters by itself (if not selected differently by the user) in an *autopilot* like mode. It constantly performs automatic restarts to escape from local solutions and to refine the current best solution. The later feature enables the algorithm to be executed even over a long time horizon, without the necessity of any user interference. The latest version of the software is well applicable on problems with up to hundreds and even thousands of optimization variables. Table 4.1 lists the problem dimension scalability intended for MIDACO based in the kind of search domain type (purely continuous (NLP), purely combinatorial/integer (IP) and mixed integer (MINLP)).

Table 4.1: Problem dimension scalability by MIDACO

Problem Type	Dimension
MINLP (mixed integer)	500
NLP (continuous)	1.000
IP (combinatorial/integer)	10.000

4.1 Reverse Communication and Distributed Computing

A key feature of MIDACO is, that it is called by reverse communication. This means, that the call to the objective function and constraints happens outside the MIDACO source code. This concept does not only guarantee a numerically stable gateway to other languages (like Matlab or MS-Excel), but also enables the software to feature a valuable option of distributed computing. Within one reverse communication step MIDACO does accept and returns an arbitrary large number of **L** iterates at once. Hence, those **L** iterates can be evaluated regarding their objective and constraint functions in parallel, outside and independently from the MIDACO source code. This idea of passing a block of **L** iterates at once within one reverse communication step to the optimization algorithm is

taken from the code NLPQLP by Schittkowski [41].

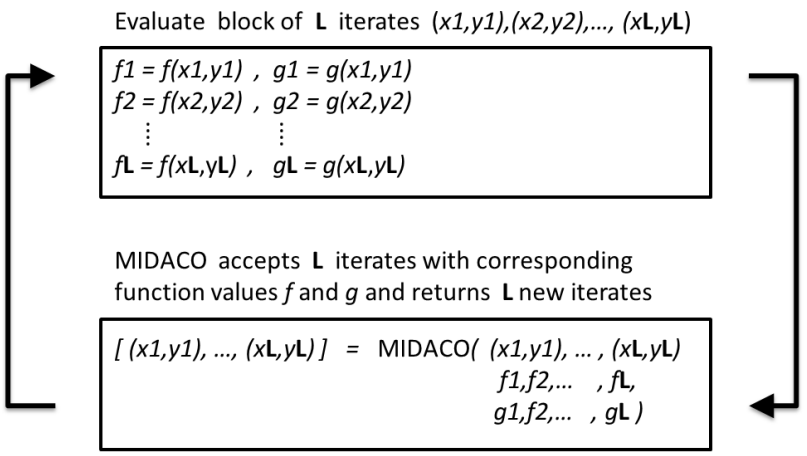

Figure 4.1: The reverse communication loop over a block of **L** iterates (x, y)

Figure 4.1 illustrates the essential reverse communication loop over the function evaluation calls to $f(x, y)$ and $g(x, y)$ and the MIDACO code. Within one loop, a block of **L** iterates is evaluated regarding their objective function $f(x, y)$ and constraints $g(x, y)$. Then those iterates are passed together with their corresponding objective and constraint values to the MIDACO code. Within MIDACO those iterates are then processed and MIDACO calculates and returns a new block of **L** iterates that needs to be evaluated again.

This concept allows an independent and user controlled distributed computing of the objective and constraint function evaluation. In other words, the parallelization option is valid for any language on any CPU architecture without the necessity of adapting the MIDACO source code in any way. This includes in particular multi-core PC's, PC-Clusters, GPU (Graphical Processing Unit) based computation and HPC (High Performance Computing). As the parallelization factor **L** can be arbitrary large, MIDACO is absolute suitable for massive parallelization. The parallelization option is considered a valuable

feature, that enables the use of MIDACO even on very cpu time consuming problems (see Section 5.4). To the best knowledge of the author, MIDACO is so far the only available evolutionary algorithm implementation, that offers such a broad parallelization feature based on the reverse communication concept.

4.2 Parameters and Print Options

With regard to the user friendliness, MIDACO does not require any parameters to be set. For experienced users however, there are seven parameters available to adjust the software to a specific need or problem. By default, all these parameters are set to zero and MIDACO will select them internally by some *autopilot* mode. If one of those is set not equal to zero by the user, it is activated and the software will act accordingly. In the following, all seven optional parameters are described:

Seed - Parameter:

Initial seed for internal Xorshift [35] pseudo random number generator within MIDACO. The *Seed* determines the sequence of pseudo random numbers in the interval $[0, 1]$ sampled by the generator. For example $Seed = 0$ may imply a random number sequence such as $\{0.3,\ 0.9,\ 0.6,\ 0.2,\ ...\}$, while $Seed = 1$ may imply a random number sequence such as $\{0.4,\ 0.1,\ 0.8,\ 0.7,\ ...\}$. Therefore MIDACO runs using an identical *Seed*, will produce exactly the same results (executed on the same computer under identical compiler conditions). As the *Seed* may be an arbitrary integer greater or equal to zero, the user can easily generate (stochastically) different runs, using a different *Seed* parameter. The advantage of a user specified random seed is, that promising runs can easily be reproduced by knowing the applied *Seed* parameter. This is in especially useful, if a run must be stopped out of some reason and should be restarted again.

Qstart - Parameter:

This parameter allows the user to specify the quality of the starting point. If $Qstart$ is set greater than 0, the initial population of iterates (also called ants) is sampled closely around the starting point (and the best known solution further on). In particular, the standard deviation for continuous variables is set to $|x_l - x_u|/Qstart$ and the mean is set to the corresponding dimension of the starting point. For integer variables the standard deviation is set to $max\{|y_l - y_u|/Qstart,\ 1/\sqrt{Qstart}\}$ to avoid a too tight sampling. The greater $Qstart$ is selected, the more closely does MIDACO search around the starting point. This option is very useful to refine previously calculated solutions. It is important to note, that this option does not shrink the search space! The original bounds x_l, y_l and x_u, y_u are still valid, only the populations of ants are specifically focused within these bounds. Further note, that $Qstart$ can also be used to $focus$ the algorithmic search of MIDACO, because the reduced standard deviations are not only applied to the starting point, but also to any further best known solution found by MIDACO.

Autostop - Parameter:

This parameter activates an internal stopping criteria for MIDACO. While it is recommended, that the user will run MIDACO for a fixed time or evaluation budget, this option allows the software to stop the optimization run by itself. *Autostop* defines the amount of internal restarts in sequence, which did not reveal an improvement in the objective function value. The greater *Autostop* is selected, the higher the chance of reaching global optimality, but also the longer the optimization run. As *Autostop* can be selected any integer greater or equal to zero, it gives the user the freedom to compromise between global optimality and cpu run time to his or her specific needs.

Oracle - Parameter:

This parameter specifies a user given oracle parameter Ω (see Chapter 3) to the penalty function within MIDACO. If *Oracle* is selected not equal to zero, MIDACO will use the *Oracle* as long as a better feasible solution has been found. In that case the regular oracle update (see Subsection 3.2.4) starts to take place. This option can be useful for problems with difficult constraints where some background knowledge on the problem exists.

Ants - Parameter:

This parameter fixes the amount of individuals (also called ants, see Definition 2.3) within a generation. This option can be useful to adapt MIDACO to expensive cpu time problems, or problems with many variables (e.g. more than hundred).

Kernel - Parameter:

This parameter fixes the kernel $\mathcal{K}$ (Definition 2.4), defining the amount of individuals saved in the solution archive $\mathcal{S}$ (Definition 2.4). The *Kernel* parameter must be used in combination with the *Ants* parameter and is intended to make MIDACO more efficiently on specific problems.

Character - Parameter:

This parameter activates a specific set of MIDACO internal parameters specially tuned for either purely continuous, purely combinatorial or mixed integer problems. If *Character* is set to zero, MIDACO will select the according set of internal parameters solely based on the problem dimensions n and n_{int}. The intention of this option is to allow the user to manually activate a different set, if a problem has a specific structure. For example, consider a mixed integer problem with 99 continuous variables and only one integer variable. In such a case it might be more promising to activate the internal *Character* for purely continuous, rather than the one for mixed integer, problems.

For the sake of a maximal portability and efficiency, the MIDACO Fortran77 and C source code does not include any printing commands by itself. Printing options are available by external subroutines freely distributed with example calls of MIDACO in different languages. Those routines allows the user to specify the printing to his or her specific needs with high accuracy. The routines especially feature the option to print the current best solution to a file in a user defined frequency of function evaluation. In other words, the user has access to the current best solution vector at any time during the optimization process. This is an important feature for applications, that are executed over a long time horizon (where the run might get corrupted and the solution would get lost otherwise).

4.3 Hybridization with SQP

In Section 6.1 and Section 6.6 MIDACO has been used in a coupled approach together with an SQP algorithm (SQP-Filtertoolbox by Prof. Gerdts, http://www.unibw.de/lrt1/gerdts/software). Here the details on this hybridization are illustrated. The most straight forward approach was applied, which is the splitting of the optimization process into two separate steps: First MIDACO is applied on the MINLP problem using the lower bounds ($x_{lower},\ y_{lower}$) as starting point, second SQP is applied on the fixed NLP problem (similar to Equation 1.2 from Section 1.2) using the best solution revealed by MIDACO as starting point. Note that MIDACO introduces a starting point (x_0, y_0) as first individual (Ant_1) within the first generation ($\mathbf{G}^1$) of its algorithmic procedure (see the example calculation of ACO in Subsection 2.4.1). Equation (4.2) states the MINLP

problem supposed for MIDACO considering a specific starting point (x_0, y_0):

$$\begin{aligned}
&\text{Minimize} \quad f(x,y) \qquad\qquad (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{Z}^{n_{int}}),\\
&\\
&\text{subject to:}\ \ g_i(x,y) \ = \ 0, \quad (i \ = \ 1,...,m_e),\\
&\qquad\qquad g_j(x,y) \ \geqslant \ 0, \quad (j \ = \ m_e+1,...,m),\\
&\qquad\qquad\quad x_{lower} \leqslant x \leqslant x_{upper}, \quad (x_{lower}, x_{upper} \in \mathbb{R}^{n_{con}} : x_{lower} \leqslant x_{upper}),\\
&\qquad\qquad\quad y_{lower} \leqslant y \leqslant y_{upper}, \quad (y_{lower}, y_{upper} \in \mathbb{Z}^{n_{int}} : y_{lower} \leqslant y_{upper}),\\
&\quad \text{and starting point } (x_0, y_0): \quad x_0 = x_{lower}, \quad y_0 = y_{lower}.
\end{aligned} \tag{4.2}$$

Let $(x^*_{midaco},\ y^*_{midaco})$ be the solution revealed by MIDACO to the original MINLP (4.2). Then SQP is applied on the fixed NLP (4.3), where the integer parameters y are fixed to the integer solution variables y^*_{midaco} given by MIDACO and the continuous starting point x_0 equals the continuous solution variables x^*_{midaco} given by MIDACO.

$$\begin{aligned}
&\text{Minimize} \quad f^y(x) \qquad\qquad (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{Z}^{n_{int}}),\\
&\\
&\text{subject to:}\ \ g^y_i(x) \ = \ 0, \quad (i \ = \ 1,...,m_e),\\
&\qquad\qquad g^y_j(x) \ \geqslant \ 0, \quad (j \ = \ m_e+1,...,m),\\
&\qquad\qquad\quad x_{lower} \leqslant x \leqslant x_{upper}, \quad (x_{lower}, x_{upper} \in \mathbb{R}^{n_{con}} : x_{lower} \leqslant x_{upper}),\\
&\qquad\qquad\qquad\ y = y^*_{midaco} \ \in \mathbb{Z}^{n_{int}},\\
&\quad \text{and starting point } (x_0): \quad x_0 = x^*_{midaco} \in \mathbb{R}^{n_{con}}.
\end{aligned} \tag{4.3}$$

Chapter 5

Numerical Results on MINLP Benchmark Sets

In this chapter extensive numerical results by MIDACO (see Chapter 4) on comprehensive MINLP benchmark sets (with up to 100 test problem instances) are presented. This is done to demonstrate the usefulness and potential of the new algorithms developed within this thesis (in esp. the mixed integer extended ACO algorithm in Chapter 2, and the oracle penalty method in Chapter 3) in the general area of MINLP. The first section of this chapter investigates in particular the performance of the oracle penalty method in comparison with commonly used constraint handling techniques used in stochastic algorithms. The second and third section of this chapter evaluates the MIDACO performance on a set of 100 (resp. 66) non-convex MINLP problems and compares the results to three established deterministic MINLP solvers (namely MISQP, BONMIN and COUENNE). The fourth and last section of this chapter explores the impact of (massive) parallelization, that is an optional feature of the MIDACO software.

This chapter shows, that the oracle penalty method is fully competitive with commonly used advanced penalty approaches in stochastic algorithms, while offering the significant advantage of only one required parameter. On the comprehensive test bed of 100 (resp. 66) MINLP benchmarks, MIDACO does clearly outperform established MINLP software in respect to the amount of global optimal solutions found. Furthermore it can be seen that MIDACO is competitive or even outperforms the other solvers in terms of cpu runtime performance. The significant benefit of parallelization is demonstrated at last, were it can be shown that MIDACO can even be competitive to deterministic algorithms in respect to the number of function evaluation (if massive parallelization is available).

5.1 Evaluation of the Oracle Penalty Method

A set of 60 constrained MINLP benchmark problems from the open literature is considered to compare the performance of the here proposed oracle penalty method (see Chapter 3) with commonly used penalty functions in evolutionary algorithms. Details on all benchmark problems can be found in the Appendix 6.6.5 in Table 25.

As penalty functions we consider besides the extended oracle penalty function (3.7) the static, death and adaptive one (see Table 3.2). For the numerical test different parameter setups for some penalty functions have been applied. Table 5.1 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 Subsection 3.2.4. 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 [10].

Table 5.1: Penalty functions and their parameters considered for numerical results

Abbreviation	Penalty function	Section	Parameters
oracle_{update}	Extended oracle	3.2.3	$\Omega^1 = 10^9$, $\Omega^{2,3,\cdots}$ updated
oracle_{fix}	Extended oracle	3.2.3	$\Omega = 10^9$
static	Static	3.1	$K = 10^9$
death	Death	3.1	none
adaptive_1	Adaptive	3.1	$\lambda(1) = 100,\ \beta_1 = 1,\quad \beta_2 = 2,\quad k = 20$
adaptive_2	Adaptive	3.1	$\lambda(1) = 50,\quad \beta_1 = 1.5,\ \beta_2 = 2.5,\ k = 10$
adaptive_3	Adaptive	3.1	$\lambda(1) = 200,\ \beta_1 = 2,\quad \beta_2 = 3,\quad k = 40$

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 $f(x^*, y^*)$ (see Table 25). If a feasible solution x with an objective function value $f(x, y)$ was found, so that:

$$\frac{|f(x,y) - f(x^*,y^*)|}{f(x^*,y^*)} \leqslant acc, \tag{5.1}$$

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 2).

The overall performance of all tested penalty functions on the set of 60 problems is displayed in Table 5.3, where Table 5.2 explains the abbreviations used. Detailed results on every problem and every penalty function can be found in the Appendix 6.6.5 in Table

29, where Table 27 explains the abbreviations used.

Table 5.2: Abbreviations for Table 5.3

Abbreviation	Explanation
Penalty	Penalty function used in MIDACO
Optimal (out of 60, [%])	Number of problems where at least one out of 100 runs a global optimal solution was found.
Feasible (out of 60, [%])	Number of problems where at least one out of 100 runs a feasible solution was found.

Table 5.3: Overall performance of different penalty methods

Penalty	Optimal (out of 60, [%])	Feasible (out of 60, [%])
Oracle_{update}	56 [0.93]	60 [1.00]
Oracle_{fix}	48 [0.80]	60 [1.00]
Static	50 [0.83]	60 [1.00]
Death	40 [0.67]	57 [0.95]
Adaptive_1	48 [0.80]	59 [0.98]
Adaptive_2	49 [0.82]	59 [0.98]
Adaptive_3	52 [0.87]	60 [1.00]

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 further 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 Subsection 3.2.3), 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 6.6.5, Table 29).

5.2 MIDACO Performance Comparison with MISQP

This section presents numerical results obtained by MIDACO on a set of 100 non-convex MINLP benchmark problems from the open literature. The set of benchmark problems is provided in Fortran by Schittkowski [40] and can be downloaded at

http://www.math.uni-bayreuth.de/~kschittkowski/mitp_coll.htm

Note, that many of these problems are originally taken form the GAMS library MINLPlib [22]. The dimension of these problems range between 2 and 100. The number of nonlinear constraints range between 0 to 54 with up to 17 equality constraints. The problems are either mixed integer or purely combinatorial problems. In conclusion, this set provides a comprehensive variety of small to medium scaled MINLP problems and allows a rigorous testing of software codes written in Fortran.

This section will first summarize already published results obtained by some deterministic SQP-based algorithms on the problem set. Then, numerical results by MIDACO are presented and compared to the ones obtained by the SQP-based algorithms. The numerical

results in this contribution refer to the MIDACO using its default parameters. As initial points the lower bounds provided for every problem in Schittkowski [40] have been used in all cases. The numerical results presented here have been performed on a computer with an Intel(R) Xeon(R) E5640 CPU with 2.67GHz clock rate.

5.2.1 Performance of SQP-based Algorithms

In a recent study by Exler et al. [17] eight sophisticated implementations of SQP-based algorithms (named MISQP) for MINLP were presented and evaluated on this set of 100 benchmarks. Table 5.4 contains essential information (taken from Exler et al. [17]) on the performance of these eight algorithms. The abbreviations of Table 5.4 are as follows:

Algorithm	-	Name of the SQP-based algorithm for MINLP used in Exler et al. [17]
Optimal	-	Number of global optimal solutions obtained out of 100 problems
Feasible	-	Number of feasible solutions obtained out of 100 problems
$Eval_{mean}$	-	Average number of function evaluations over 100 problems

Table 5.4: Performance of SQP-based algorithms on 100 MINLP benchmarks

Algorithm	***Optimal***	*Feasible*	$Eval_{mean}$	$Time_{mean}$
MISQP	**89**	100	500	0.39
MISQP/bmod	**71**	100	340	0.20
MISQP/fwd	**81**	100	396	0.11
MISQP/rst0	**69**	99	241	0.14
MISQPOA	**91**	100	1,093	0.65
MISQPN	**74**	98	1,139	0.17
MINLPB4/bin	**92**	100	1,787	30.91
MINLPB4/int	**88**	94	218,881	4.11

The algorithms MISQP/bmod, MISQP/fwd and MISQP/rst0 represent different variants of the basic-MISQP algorithm. The algorithms MISQPOA and MISQPN are enhanced by an Outer Approximation method. The algorithms MINLPB4/bin and MINLPB4/int

are enhanced by Branch and Bound. For detailed information on all algorithms consult Exler et al. [17]. Determining the success of an algorithm in finding a global (or best known) optimal solution is done in Exler et al. [17] by the following criteria:

$$\frac{|f(x,y) - f(x^*,y^*)|}{|f(x^*,y^*)|} < \varepsilon, \tag{5.2}$$

where (x, y) is the (feasible) solution obtained by the algorithm, (x^*, y^*) is the best known solution and ε is some tolerance. Wether a solution is feasible or not is dependent on the L_∞-norm of the vector of constraint violations. A solution (x, y) is considered feasible, if:

$$\|g(x,y)\|_\infty < acc, \tag{5.3}$$

where acc is some accuracy. In Exler et al. [17] a tolerance of $\varepsilon = 0.01$ and an accuracy of $acc = 0.0001$ were applied for the numerical results.

All SQP-based algorithms were started from a priori defined initial points given for every problem in the source code of the library by Schittkowski [40]. In Exler et al. [17] there is no investigation on the impact of these pre-defined initial points on the performance of the eight algorithms. In other words, it is not known, if and how much the results would change, in case other (e.g. random initial points) would have been used.

5.2.2 MIDACO Performance on 100 MINLP Benchmarks

Here numerical results obtained by MIDACO 3.0 for the set of 100 MINLP problems are presented. The same criteria for global (or best known) solutions and feasibility (illustrated in equation (5.2) and (5.3)) like in Exler et al. [17] is applied. For the test runs of MIDACO a tolerance of $\varepsilon = 0.01$ and an accuracy of $acc = 0.0001$ was used,

which are identical to those tolerances as used in Exler et al. [17]. In the following, the term *global optimal* solution will always refer to the best known solution provided in Schittkowski [40].

A critical aspect of MIDACO (and most stochastic algorithms) is the stopping criteria. For the results in this section, there are two stopping criteria applied. Firstly, a maximal cpu time budget of 300 seconds (5 minutes) for every test problem and secondly the success criteria for global optimality presented in equation (5.2). This means, in both cases MIDACO is stopped from outside, it does not stop by itself. In contrary to this, the SQP-based algorithms presented in Exler et al. [17] stop by themselves and the optimality criteria is checked afterwards. An investigation of the internal stopping criteria of MIDACO, named *Autostop*, can be found in [47].

As initial point random points are used that are stochastically generated for every individual test run. Hence MIDACO does not make use of the pre-defined initial points provided in the library. As MIDACO is of stochastic nature, the full set of 100 problems is evaluated 10 times using a different random seed (from 0 to 9) for the internal pseudo random number generator. This procedure ensures objective conclusions on the robustness of the software regarding the pseudo random number generator.

Table 5.5 lists the results obtained by MIDACO by 10 runs with different *Seed* on the full set of 100 problems. Besides the number of global optimal and feasible solutions, the average number of function evaluation and time and the total required cpu time is reported. The abbreviations for Table 5.5 are as follows:

$Seed$	-	Initial seed for MIDACO's internal pseudo random number generator
Optimal	-	Number of global optimal solutions obtained out of 100 problems
$Feasible$	-	Number of feasible solutions obtained out of 100 problems
$Eval_{mean}$	-	Average number of function evaluation over global optimal solved problems
$Time_{mean}$	-	Average cpu time (seconds) over global optimal solved problems

Table 5.5: MIDACO Performance on 100 MINLP Benchmarks

$Seed$	***Optimal***	$Feasible$	$Eval_{mean}$	$Time_{mean}$
0	**96**	99	1,656,979	4.54
1	**96**	99	3,223,015	9.01
2	**96**	98	1,673,873	4.20
3	**97**	98	2,235,463	6.53
4	**96**	98	2,054,099	6.08
5	**97**	99	1,485,525	4.82
6	**95**	99	1,641,648	4.07
7	**97**	99	1,724,627	5.90
8	**95**	99	1,120,204	2.77
9	**96**	98	2,313,829	7.93

The results in Table 5.5 show a very robust MIDACO performance always obtaining between 95 and 97 global optimal solutions and between 98 and 99 feasible solutions. The average amount of function evaluation ranges between 1.5 and 3.2 million, while the average cpu time varies between 2.77 to 9.01 seconds. Note, that the MIDACO software is able to process millions of iterates within seconds on a standard computer (not taking into account function evaluation time, but only the internal MIDACO time). Regarding the random seed, the best run was obtained for $Seed = 5$, individual results for this test run can be found in Appendix B.

Comparing the results of Table 5.5 with the ones obtained by the SQP-based algorithms presented in Table 5.4, MIDACO robustly achieved a significantly higher percentage of global optimal solutions than any of the SQP-based algorithms (which range between 69 and 92 global optimal solutions). In favor of MIDACO, this conclusion must further take

into account, that the SQP-based algorithms were started only one time from pre-defined initial points. MIDACO instead was not making use of the pre-defined initial points and was tested 10 times on the full library.

Regarding the number of function evaluation, MIDACO performs significantly more evaluation than the SQP-based algorithms (which widely range between 241 and 218,881 evaluation). This results is however expected, as stochastic algorithms like MIDACO are known to require much more evaluation than deterministic ones like SQP. In terms of cpu-time performance MIDACO is at least competitive with the SQP-based algorithms (which widely range between 0.11 and 30.91 seconds). In favor of MIDACO, one has to further take into account, that the mean values for evaluation and cpu-time are calculated over 95 to 97 global solutions, while those of the SQP-based algorithms are calculated only over 69 to 92 global solutions.

5.3 MIDACO Performance Comparison with BONMIN and COUENNE

In addition to the comparison of MIDACO to the set of SQP based MINLP solvers given in Subsection 5.2.1, a further comparison to the established MINLP solvers BONMIN [6] and COUENNE [4] is given here. The solver BONMIN implements a variety of deterministic algorithms (in esp. Branch & Bound and Outer Approximation) and ensures global optimal solutions for convex problems, for non-convex MINLP problems it is a heuristic like MIDACO. The solver COUENNE is a price wining software (COIN-OR Cup 2010, see http://meetings2.informs.org/Austin2010/blog/?p=88), based on a Branch & Bound algorithm and aims at finding global optimal solutions even for non-convex MINLP

problems. Both solvers are provided by the *Computational Infrastructure for Operations Research* (COIN-OR) and are distributed within the GAMS [21] environment. These solvers have been used here *out of the box*, without any attempt to specify or tune their parameters or settings. A subset of 66 instances from the 100 MINLP benchmarks (see Table 38) is considered for evaluating purposes. This subset represent those problems from the 100 MINLP benchmarks, that are originally taken from the GAMS MINLPlib [22] benchmark library. A cpu time budget of 300 seconds has been applied to all solvers for each instance as maximal time limit. In case of MIDACO the automatic stopping criteria (see Section 4.2) was activated. A value of 50 was chosen for the *Autostop* parameter, which seems to provide a good balance between solution quality and cpu-runtime. For the deterministic solvers BONMIN and COUENNE only one test run for every problem was performed, using the pre-defined starting point from the GAMS MINLPlib. As MIDACO is a stochastic solver, 10 test runs were performed for every problem instance, using a different random seed. MIDACO did not make use of pre-defined starting points and uses the lower bounds as starting point on all instances instead.

Table 5.6 shows the comparison of the three tested solvers regarding the number of global optimal solutions (***Optimal***), the number of feasible solutions (*Feasible*) and the average and total cpu-time required. In case of MIDACO the variance of global optimal and feasible solutions, based on the different random seeds, is reported. The individual results by all three solvers on all 66 problems corresponding to Table 5.6 can be found in the Appendix 6.6.5, Table 43. Function evaluation are not reported in Table 5.6, as those information is not consistently reported within the GAMS environment for BONMIN and COUENNE. In case of MIDACO, Table 5.5 already gives evidence on the required function evaluation. All three solvers were tested on the same computer using an Intel(R) Core(TM) i7 Q820 CPU with 1.73GHz clock rate and 4GB RAM.

Table 5.6: BONMIN, COUENNE and MIDACO on 66 MINLP benchmarks

Solver	***Optimal***	*Feasible*	$Time_{aver}$	$Time_{total}$
BONMIN	**49**	64	17.99	1187.62
COUENNE	**48**	64	40.36	2664.31
MIDACO	**51 $\sim$ 62**	64 $\sim$ 65	31.41	2072.73

Based on the results in Table 5.6 it can be concluded, that MIDACO is fully competitive to the solvers BONMIN and COUENNE regarding solution quality and cpu-runtime. MIDACO outperforms both regarding the number of global optimal solutions found. In terms of cpu-runtime, MIDACO is able to outperform COUENNE, but is slower than BONMIN. Interpreting the results in Table 5.6, one has to take further into account, that testing BONMIN and COUENNE within GAMS implies a significant advantage for those solvers, as GAMS provides gradients to those algorithms for *free* (in terms of cpu-time). A further observation regarding the global optimality convergence proof hold by COUENNE should be mentioned here. COUENNE reported falsely in 10 out of 66 problems (this is 15.2%) a local solution as global optimal. This is assumably due to numerical difficulties and software bugs (e.g. scaling problems).

5.4 MIDACO Performance using Parallelization

As seen in Section 5.2.2 the MIDACO software is able to process millions of iterates within seconds (Note: This statement refers only to the internal calculation time of MIDACO and does not take into account the time for the problem function evaluation). Therefore MIDACO can achieve a very competitive cpu-time performance with a high chance of global optimality, if problem function evaluation are computationally inexpensive.

Many real world applications however, are computationally expensive. Thus, performing

millions of function evaluation in serial is not practicable. Performing them in parallel however, can be done in a reasonable time, if an adequate cpu architecture is available. The parallelization option offered by MIDACO is based on this idea. By (massively) parallelizing the problem function evaluation, even cpu time expensive problems become solvable by MIDACO in a reasonable time.

In the following, a series of experiments is performed, investigating the impact of the parallelization factor **L** (see Section 4.1) on the MIDACO performance on the same set of 100 MINLP problems known from above. A fixed budget of *evaluation blocks* is considered as budget for MIDACO. A block denotes here the amount of **L** iterates that are evaluated and passed to MIDACO within one reverse communication loop (see Figure 4.1). Regarding the computational time performance, the amount of blocks processed by MIDACO is directly comparable to the amount of serial processed function evaluation by an algorithm. For the experiment presented here, the problem function evaluation of **L** iterates given in every block were executed in serial, rather than actually parallelized. As due to the reverse communication concept the function evaluation are completely independent of the MIDACO code, this makes absolutely no difference for MIDACO. Hence those experiments only simulate the impact of parallelization on the MIDACO performance. Note that nevertheless, the conclusions on the impact of an actual parallelization are absolute accurate and valid.

A series of test runs considering a maximal budget of 100,000 evaluation blocks is performed. This budgets is chosen to express the scenario of a cpu time expensive application, where not more than 100,000 (serial processed) function evaluation can be executed in a reasonable time. The parallelization factor **L** will be increased stepwise from 1 to 50,000. As done in Section 5.2.2, the success criteria (5.2) is applied to stop MIDACO, in case it reveals the global optimal solution before performing the maximal budget of blocks. Table

5.7 lists the number of global optimal and feasible solutions regarding the corresponding parallelization factor **L**. The average number of blocks to be evaluated and processed by MIDACO is given in addition. The abbreviations for Table 5.7 are as follows:

L	-	Parallelization factor for MIDACO (see Section 4.1)
Optimal	-	Number of global optimal solutions obtained out of 100 problems
$Feasible$	-	Number of feasible solutions obtained out of 100 problems
$Block_{mean}$	-	Average number of evaluation blocks over global optimal solved problems

Table 5.7: Impact of **L** given a maximal budget of 100,000 blocks

L	***Optimal***	$Feasible$	$Block_{mean}$
1	**61**	90	7,736
5	**69**	92	2,118
10	**75**	91	4,394
50	**80**	95	3,429
100	**80**	98	2,406
500	**82**	98	1,161
1,000	**83**	98	2,458
5,000	**84**	98	2,203
10,000	**86**	98	2,100
50,000	**89**	98	1,916

Note, that a different parallelization factor **L** implies a different stochastic behavior of MIDACO. This explains the non-monotonic variations in average number of evaluation blocks.

The results presented in Table 5.7 demonstrate the significant impact of the parallelization factor **L** on the MIDACO performance. Like expected, the results corresponding to a parallelization factor $\mathbf{L} = 1$ (which means no parallelization at all) are very weak with only 61 global optimal solved problems and an average of 7,736 evaluation blocks. However, assuming a massive parallelization factor of $\mathbf{L} = 50{,}000$, the number of global optimal solved problems can be increased up to 89 with a corresponding average number of 1,916

evaluation blocks. As the number of blocks directly corresponds with the number of serial processed function evaluation, it can be concluded that MIDACO can even be competitive with some of the SQP-based algorithms (see Table 5.4) in terms of function evaluation, if (massive) parallelization capabilities are available.

Chapter 6

Numerical Results on Real World Applications

In this chapter numerical results obtained by MIDACO (see Chapter 4) on real world applications are presented. In compliance with the topic of this thesis, the focus here is only on (aero)space applications. Nevertheless, the usage of MIDACO is by no means limited to this area and several applications on real world applications from other fields have been performed and published (see [43], [44] and [28] for example). Table 6.1 gives an overview on real world applications solved by MIDACO, regarding their academic area and the solution quality achieved by MIDACO. Table 6.1 uses the following abbreviations:

Area	-	Industrial or academic area of the corresponding application
Name	-	Name of the application used in the *Reference*
Solution	-	Quality of the solution obtained by MIDACO
n	-	Number of decision variables of the application
Reference	-	Literature or (web-) reference for the application
Section	-	Section in this chapter, that refers to this application
n.a.	-	*not available*

Table 6.1: Real world applications solved by MIDACO

Area	*Name*	***Solution***	n	*Reference*	*Section*
Aerospace	F8-Aircraft	**best**	6	[45]	6.1
(Aero)Space	Heat Shield	***new* best**	31	[43]	6.2
Space	Satellite	*first solution*	5	[49]	6.3
Space	ESA/ACT GTOP	**(*new*) best**	6∼26	[46]	6.4
Space	Space Mission	**best**	21	*n.a.*	6.5
Space	Launcher	***new* best**	128	*n.a.*	6.6
Chem. Eng.	TEP	**best**	43	[44]	*n.a.*
Chem. Eng.	WWT.COST.1	**best**	4	[44]	*n.a.*
Chem. Eng.	WWT.COST.2	**best**	8	[44]	*n.a.*
Chem. Eng.	WWT.COST.3	**best**	13	[44]	*n.a.*
Chem. Eng.	TP4	***new* best**	52	[45]	*n.a.*
Chem. Eng.	TP5	***new* best**	113	[45]	*n.a.*
Robotics	Camera	*first solution*	45	[28]	*n.a.*

The following sections illustrate all the (aero)space applications listed in Table 6.1 in detail. While the MIDACO usage on most of these applications have been already published in the open literature, the Space Mission (Section 6.5) and Launcher (Section 6.6) application here are the latest ones and their separate publication is currently in preparation. Note that the numerical results presented here for the F8-Aircraft (Section 6.1) application differ from the already published results in [45], as significant improvements could be achieved meanwhile on this problem. In case of the F8-Aircraft (Section 6.1) and Launcher (Section 6.6), the application was solved by a hybrid combination of MIDACO and a SQP algorithm, were the later was used for refinement purposes.

In total, the results summarized in Table 6.1 demonstrate well the real-world capabilities of the MIDACO software (which is based on the theoretical algorithms developed in this thesis). In particular MIDACO is able to obtain the best known or even **new best known** solutions in any of the listed applications.

Note that the implementations of the applications discussed in Section 6.1, 6.5 and 6.6 can be downloaded at http://www.midaco-solver.com/applications.html.

6.1 Optimal Control of an F8-Aircraft Manoeuvre

In this subsection the optimal control of an aircraft manoeuvre is discussed. This application is known as the F-8 aircraft control problem introduced by Kaya and Noakes [32]. Here we refer to a formulation of this application that is available from *mintOC* [39] at http://mintoc.de/index.php/F-8_aircraft. The concrete implementation used to generate the following results can be downloaded at http://www.midaco-solver.com/applications.html. Several reference solutions to this application can be found at the *mintOC* [39] webpage. Among those are solutions obtained by well known solvers such as *BONMIN*, *KNITRO* and *IPOPT*. Note, that none of the above mentioned solvers is capable to solve this application to its current best known solution, due to the information given on *mintOC* [39] and that those solvers might have considered different reformulations of the problem (e.g. NLP reformulations). Hence we consider this as a challenging application with good possibilities to compare the solution quality obtained by the here considered approach with concurrent ones.

The objective of this application is the minimization of the final time of a simplified aircraft manoeuvre. The manoeuvre is performed using a bang-bang control approach, thus the control can only switch between two values. Here this application is formulated as a NLP optimal control problem, applying 6 different stages, whereas every stage represents a switch in the bang-bang control. This implies 6 optimization variables, which correspond to the starting time points of the stages, defining the switching time point of the bang bang control. This approach coincides with the representation of the reference solutions

given at *mintOC* [39], where the optimal control stages are referred to as arcs. Besides the 6 optimization variables, 3 equality constraints must be fulfilled at the end of the manoeuvre, representing the correct final state of the aircraft. This is in especially that all three differential states must be zero at the final time of the manoeuvre.

Equation 6.1 expresses the F8 aircraft manoeuvre regarding the three time depended considered differential states $\dot{x}_0$, $\dot{x}_1$ and $\dot{x}_2$. The initial state $x(0)$ and the final state $x(t_{final})$ are also stated, whereas the final state implies the above mentioned three equality constraints.

$$
\begin{aligned}
\text{Minimize} \quad & t_{final}, \\
\text{s.t.:} \quad \dot{x}_0 = & -0.877x_0 + x_2 - 0.088x_0x_2 + 0.47x_0^2 - 0.019x_1^2 - x_0^2x_2 + 3.846x_0^3, \\
& - (0.215\xi - 0.28x_0^2 - 0.47x_0\xi^2 - 0.63\xi^3)w, \\
& - (-0.215\xi + 0.28x_0^2\xi - 0.47x_0\xi^2 + 0.063\xi^3)(1-w), \\
\dot{x}_1 = & \; x_2, \\
\dot{x}_2 = & -4.208x_0 - 0.396x_2 - 0.47x_0^2 - 3.564x_0^3, \\
& - (20.967\xi - 6.265x_0^2\xi - 46x_0\xi^2 - 61.4\xi^3)w, \\
& - (-20.967\xi + 6.265x_0^2\xi - 46x_0\xi^2 + 61.4\xi^3)(1-w), \\
x(0) = & \; (0.4655, 0, 0)^T, \\
x(t_{final}) = & \; (0,0,0)^T.
\end{aligned}
\tag{6.1}
$$

The bang-bang control $w \in \{0,1\}$ is defined in Equation 6.2. The bang-bang structure is here defined by the time points $t_1,\ t_2,\ t_3,\ t_4,\ t_5$ and t_{final}, marking the beginning or end of a different stage (also called *arc*). These six time points are the decision variables,

whereas the last one (t_{final}) also expresses the objective function.

$$w(t) = \begin{cases} 1 & \text{, if } 0 \leqslant t < t_1, \\ 0 & \text{, if } t_1 \leqslant t < t_2, \\ 1 & \text{, if } t_2 \leqslant t < t_3, \\ 0 & \text{, if } t_3 \leqslant t < t_4, \\ 1 & \text{, if } t_4 \leqslant t < t_5, \\ 0 & \text{, if } t_5 \leqslant t \leqslant t_{final}. \end{cases} \tag{6.2}$$

The F8-Aircraft application is solved by a combination of MIDACO (see Chapter 4) and a SQP algorithm (SQP-Filtertoolbox by Prof. Gerdts). MIDACO is first applied on the optimal control problem using the lower bounds (zero) as starting point, whereas the SQP algorithm is then afterwards called, applying the MIDACO solution as starting point. Two different setups for MIDACO are assumed: i) using default parameters and ii) using tuned parameters regarding the algorithm (in esp. *Qstart* and *Oracle*, see Section 4.2). For the **default setup**, MIDACO is given a maximal time budget of 600 seconds, or stops before this limit by its own automatic stopping criteria (in esp. *Autostop* = 5, see Section 4.2). For the **tuned setup**, MIDACO is given a maximal time budget of 60 seconds, a *Qstart* parameter of 100 (in order to focus the search process)and an *Oracle* parameter of 4.0 (which is based in the best known solution corresponding to an objective function value of 3.78, see Table 6.4). MIDACO assumes a moderate accuracy of 10^{-2} for the constraint violation, whereas the SQP algorithm assumes a higher accuracy of 10^{-6}. The idea behind this approach is that MIDACO delivers a reasonable good starting point for the SQP algorithm, which then returns a highly accurate solution. Table 6.2 and Table 6.3 shows the numerical results by this approach for 10 test runs (using a different random seed each time) by MIDACO using default parameters and tuned ones respectively. All runs were performed on a computer with an Intel(R) Core(TM) i7 Q820 CPU with 1.73GHz clock rate and 4GB RAM.

Table 6.2: Results of 10 test runs on F8-Aircraft using MIDACO (**default**) and SQP

	MIDACO			SQP		
Test Run	Objective	Evaluation	Time	Objective	Evaluation	Time
1	4.017191	575038	127.34	3.780211	106	0.04
2	3.753558	467211	126.49	3.780212	184	0.09
3	6.830677	313259	82.77	6.827373	221	0.13
4	3.740785	904630	262.50	3.780211	210	0.09
5	6.873262	1145671	304.39	6.827373	299	0.18
6	3.735261	422985	101.00	3.780211	73	0.03
7	6.860254	2097135	568.07	3.780211	711	0.44
8	3.766495	329588	89.35	3.780211	313	0.14
9	6.288805	1239587	600.00	6.322984	104	0.06
10	4.573276	667050	227.74	3.780211	3274	1.95
Average:	5.043956	820663	248.96	4.643921	549	0.32

Table 6.3: Results of 10 test runs on F8-Aircraft using MIDACO (**tuned**) and SQP

	MIDACO			SQP		
Test Run	Objective	Evaluation	Time	Objective	Evaluation	Time
1	3.775308	107930	60.00	3.780211	203	0.13
2	3.781723	133881	60.00	3.780211	170	0.07
3	3.735079	109770	60.00	3.780211	63	0.04
4	3.918386	104279	60.00	3.780211	206	0.14
5	4.092283	102022	60.00	3.780211	207	0.14
6	4.020482	107319	60.00	3.780211	92	0.06
7	3.733805	114478	60.00	3.780211	63	0.04
8	3.797692	100179	60.00	3.780211	157	0.11
9	4.048429	101560	60.00	3.780212	316	0.20
10	3.786220	121182	60.00	3.780211	108	0.08
Average:	3.868941	110260	60.00	3.780211	158	0.10

Table 6.2 displays the best solution found by the combination of MIDACO and SQP from Table 6.4 and compares it with the best known solution (found by Sager) from *mintOC* [39].

Table 6.4: F-8 Aircaft control problem solutions

Arc	w(t)	Sager	MIDACO + SQP
1	1	1.13492	1.1368996475
2	0	0.34703	0.3457308852
3	1	1.60721	1.6071985027
4	0	0.69169	0.6048388008
5	1	0	0.0000000000
6	0	0	0.0855435144
Infeasibility	-	2.21723e-07	0.4896e-13
Objective	-	3.78086	3.780211

Figure 6.1 displays the three differential states of the aircraft model over time, corresponding to the best solution presented in Table 6.4.

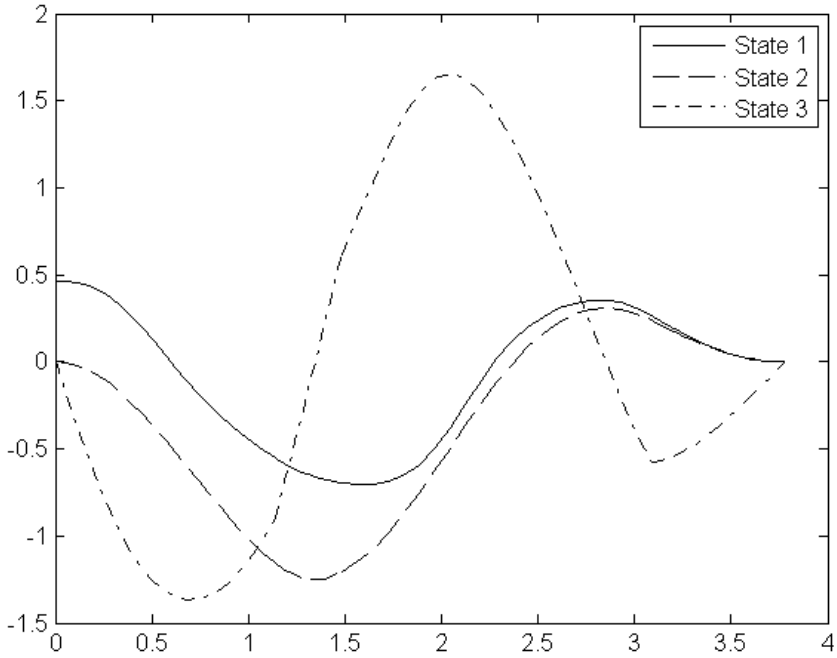

Figure 6.1: Differential states corresponding to best known solution

From the results from Table 6.2 it can be seen, that MIDACO (using **default** parameters) delivers four out of ten times a solution close to the best known one. In two cases MIDACO

delivers a moderate solution quality of an objective function value between 4.01 and 4.57, while in the remaining four cases a worse objective function value between 6.28 and 6.87 is revealed. The SQP algorithm is able to refine the MIDACO solutions to the best known one (corresponding to an objective function value of 3.78) in seven out of ten cases to a high accuracy in less than a second.

From the results from Table 6.3 it can be seen, that MIDACO (using **tuned** parameters) delivers in all cases good solutions (in especially those with an objective function value smaller or equal to the predefined $Oracle = 4.0$). Note that the cpu time budget for MIDACO in Table 6.3 is only 60 instead of 600 seconds like in Table 6.2! The SQP algorithm is able to refine the MIDACO solutions to the best known one in all cases. Comparing the MIDACO results from Table 6.2 and Table 6.3 demonstrate the possible performance gain in tuning the algorithmic parameters and the effectiveness of the oracle penalty method (see Chapter 3).

In total it can be concluded, that the proposed hybrid approach here is well capable to robustly solve this application in a reasonable time with high accuracy.

6.2 Thermal Insulation System Application (Heat Shield Problem)

In this subsection the optimization of a load-bearing thermal insulation system is described. A thermal insulation system is characterized by hot and cold surfaces with a series of heat intercepts and insulators between them. It is assumed that the insulators act as a mechanical support, a system with such a property is called load-bearing. These kind of systems find its application for example in space borne magnets or the heat shield

thermal protection system of space crafts. Here a specific model ('Heat Shield Problem Files') obtained from Abramson [2] is considered. The aim of the optimization is to minimize the required power (P), which is needed to maintain a stable temperature for the surfaces and intercepts used within the system. Details on the thermal insulation system can be found in Abramson [1].

While in Abramson [1] the optimization problem was stated as a mixed variable problem (MVP) with a variable number of intercepts, here we have selected a MINLP formulation with a fixed number of intercepts. Based on the solution provided in Abramson [2] the number of intersections was fixed to eleven. Consequently our MINLP formulation consists of 20 continuous and 11 integer decision variables, plus 2 nonlinear inequality constraints. Regarding the lower and upper bounds for the continuous variables, the solution values provided in Abramson [2] were used once again as a reference. We assumed bounds of 50 percent around those values for the real variables. Regarding the integer variables, every integer represents an insulator material used in an intercept, and 7 different types of materials were considered in Abramson [1], so these integer variables have a range from 1 to 7, with the materials being nylon, teflon, epoxy normal, epoxy plane, aluminum, carbon-steel and steel.

To illustrate the multimodality, and therefore the complexity of this problem, a frequency histogram for the feasible solutions found by a multistart execution of the local solver MISQP [18] with 100 random initial points is shown in Figure 6.2.1 (only solutions with an objective function value lower than 1000 are displayed). Out of this 100 runs of MISQP, 54 did not converge to a feasible solution. Of the 46 runs that converged to a feasible solution, the best objective function value found was 141.35 (P). In total, the multistart procedure required 58325 objective function evaluations. Note, that for the sake of comparison an evaluation limited MISQP multistart with random initial points

is considered beneath in addition to this multistart with a fixed number of 100 MISQP executions.

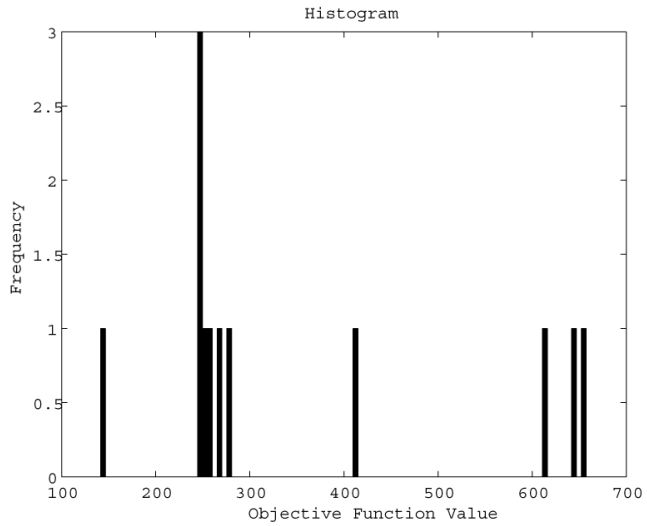

Figure 6.2: Frequency histogram of feasible solutions for the 'Heat Shield Problem'

For the optimization of this application, the software code ACOmi is considered here. ACOmi is a Matlab implementation of the extended ACO algorithm described in Chapter 2 in combination with the oracle penalty method presented in Chapter 3 and offers the hybridization with the local solver MISQP. Details on the ACOmi software can be found in [43]. ACOmi is considered the first beta version of the MIDACO software, described in Chapter 4. The ACOmi solver is considered here in three different setups regarding a hybridization with the local solver MISQP. Table 6.5 lists the different setups with details on the hybridization.

Table 6.5: ACOmi setups regarding local solver MISQP

Setup	Description of the hybridization level
`ACOmi`$_1$	No hybridization with MISQP
`ACOmi`$_2$	MISQP is frequently called within ACOmi after every ACO generation
`ACOmi`$_3$	MISQP is called only once, using the ACOmi solution as starting point

The performance of ACOmi is compared to MITS [16], which is an implementation of a Tabu Search [25] algorithm for MINLP using MISQP as a local solver, and the evaluation limited multistart of MISQP with random initial points. A maximal budget of 10000 function evaluation was set for every solver and 30 test runs have been performed. A precision of 10^{-4} regarding the l^1 Norm over all constraint violations and the objective function value was claimed.

In addition to our own obtained results by MITS, the MISQP multistart and ACOmi, two solutions presented in Abramson [1] were taken as references: the NOMADm solution obtained by Abramson [1], which is identical to the above mentioned reference solution provided by Abramson [2], and another solution by Kokkolaras et al. [33]. While in Abramson [1] only normalized values of the objective function were reported, we have focused on the original (not normalized) values corresponding to the power (P). The normalization is done by a multiplication of the power with the system-load (L) and a division by a cross-sectional area (A). Using these normalized values, the NOMADm solution was 23.77 ($\frac{PL}{A}$), and the Kokkolaras one 25.29 ($\frac{PL}{A}$). Using the 'Heat Shield Problem Files' provided by Abramson [2] we obtained a not normalized NOMADm solution value of 106.35 (P), which is consistent with the convergence curve performance of the pure power (P) presented in Abramson [1].

The results obtained by MITS, the evaluation limited MISQP multistart and the ACOmi setups are given in Table 6.6, where the best (f_{best}), worst (f_{worst}) and mean (f_{mean}) (feasible) objective function value are reported. All 30 test runs of all solvers converged to feasible solutions. In addition the mean number of function evaluation ($\text{eval}_{\text{mean}}$) and the corresponding cpu-time ($\text{time}_{\text{mean}}$) in seconds is also reported for all solvers.

Table 6.6: Results for the Heatshield problem

solver	feasible	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
$\texttt{ACOmi}_1$	30	105.86	110.01	107	10051	206.03
$\texttt{ACOmi}_2$	30	105.79	163.54	112.9	10370	331.54
$\texttt{ACOmi}_3$	30	105.82	121.35	107.74	10375	216.61
MISQP	30	154.6	9427	581.91	10202	281.89
MITS	30	111.04	150.46	124.71	11037	518.52

In addition to Table 6.6, Table 6.7 gives detailed information on the best decision vectors (x^*, y^*) obtained by MITS and ACOmi (best solution obtained by setup $\texttt{ACOmi}_2$). Also the above mentioned NOMADm solution and the solution values (original and normalized) are given.

Table 6.7: Best solution (x^*, y^*) by NOMADm, MITS and ACOmi

solution information	$(x^*, y^*)_{NOMADm}$	$(x^*, y^*)_{MITS}$	$(x^*, y^*)_{\texttt{ACOmi}_2}$
continuous variables x^*:			
1	0.625	0.843	0.321
2	8.125	6.670	7.658
3	7.968	9.652	8.639
4	7.812	11.652	7.153
5	12.344	6.172	13.781
6	26.094	15.854	25.698
7	8.125	12.187	8.237
8	5.312	7.968	5.780
9	5.000	7.500	5.087
10	5.625	7.373	6.613
11	4.250	4.201	4.200
12	7.737	6.401	7.294
13	12.369	11.788	12.089
14	18.094	20.795	17.177
15	29.912	25.722	30.185
16	71.094	47.717	71.257
17	105.940	71.000	107.266
18	135.470	114.119	139.299
19	165.940	165.459	171.534
20	202.030	206.272	214.485
categorical variables y^*:			
1	Epoxy normal	Carbon steel	Nylon
2	Epoxy normal	Epoxy normal	Epoxy normal
3	Epoxy normal	Epoxy normal	Epoxy normal
4	Epoxy normal	Epoxy normal	Epoxy normal
5	Epoxy normal	Epoxy normal	Epoxy normal
6	Epoxy normal	Epoxy normal	Epoxy normal
7	Epoxy normal	Epoxy normal	Epoxy normal
8	Epoxy normal	Epoxy normal	Epoxy normal
9	Epoxy normal	Epoxy normal	Epoxy normal
10	Epoxy normal	Epoxy normal	Epoxy normal
11	Epoxy normal	Epoxy normal	Epoxy normal
solution value:			
P (original)	106.355	111.043	105.787
$\frac{PL}{A}$ (normalized)	23.768	24.815	23.641

The convergence curves of all 30 test runs by MITS and the `ACOmi`$_1$ setup are given in Figure 6.3. Note that the plots uses double logarithmic scale.

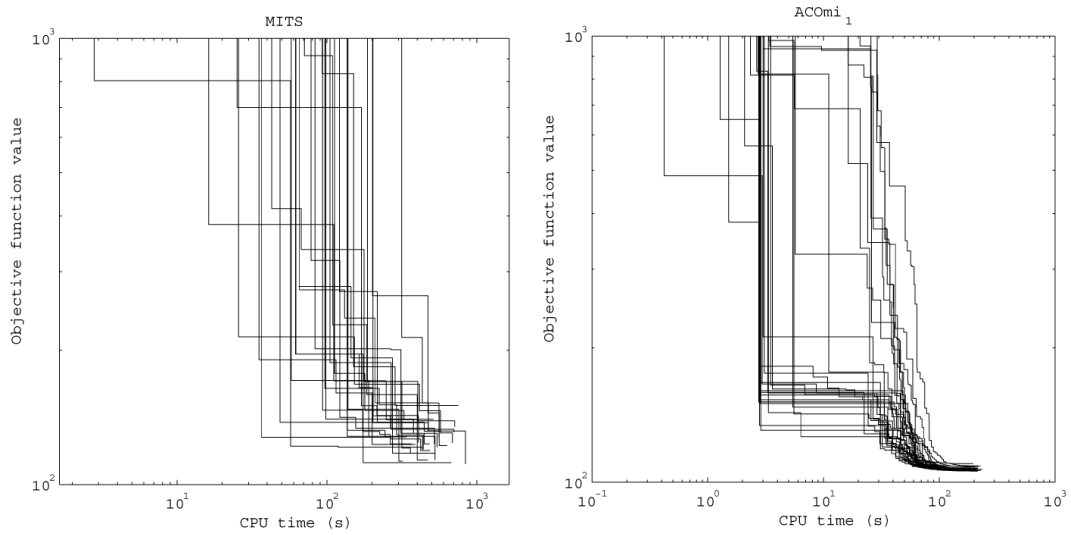

Figure 6.3: Convergence curves of MISQP and `ACOmi`$_1$ for the Heatschield problem

Analyzing the results of `ACOmi` on this application and comparing them with those of MITS and the MISQP multistart, `ACOmi` was clearly outperforming both. In addition the best results by all three `ACOmi` setups are slightly better than the best solution found by NOMADm (see Abramson [1]). It is significant, that the mean objective function value of around 107 obtained by `ACOmi`$_1$ and `ACOmi`$_3$ is much better than the one obtained by `ACOmi`$_2$, which is the `ACOmi` setup calling the local solver after every generation. As the MISQP multistart achieved only a pure best objective function value of 154.6 and a much worse mean value of 581.91, the use of this local solver for this application does not seem promising. This is also assumed as an explanation why MITS performed significantly worse than `ACOmi` on this application. It seems that the performance of MITS is heavily dependent on the local solver MISQP.

6.3 Satellite Constellation Optimization

This subsection describes the application of the MIDACO software (see Chapter 4) on a space based situational awareness application. Note, that this application was undertaken independently from the author of this thesis by Takano and Marchand [49] at the Department of Aerospace Engineering of the University of Texas at Austin (USA) in 2011. However, as this application fits the topic of this thesis very well, it should be briefly presented here based on the description given in Takano and Marchand [49].

This application considers the optimization of coverage capabilities by satellites elliptical orbiting around the earth. Two distinct orbits are considered, employing a variable number of satellites. The objective function is to minimize the total number of satellites in this two orbit system while fulfilling a continuous coverage constraint. The application is imposed as a MINLP problem, considering two integer variables representing the number of satellites on each orbit. Furthermore, the semi-major axis and eccentricity of the orbits is considered as continuous optimization variables. Table 6.8 shows the starting point and solution obtained by MIDACO after 300 seconds (corresponding to 682 function evaluation).

Table 6.8: Starting Point and MIDACO Solution

Variable	Description	Starting Point	MIDACO Solution
a	semi-major axis (both orbits)	10000 km	8588 km
e	eccentricity (both orbits)	0.25	0
n_1	number of satellites on orbit 1	3	0
n_2	number of satellites on orbit 2	3	6
	Objective Function Value:	*infeasible*	6

MIDACO did reveal a first feasible solution corresponding to an objective function value

of 12 after 15 function evaluation and converged to the presented solution with an objective function value of 6 after 98 function evaluation. Based on the information given in Takano and Marchand [49], the solution revealed by MIDACO is the expected one for this application.

6.4 ESA/ACT: Global Trajectory Optimization Problems

Here a set of (constrained) continuous optimization benchmark problems provided by the *Advanced Concept Team* (ACT) of the *European Space Agency* (ESA) is considered. This benchmark set is known as the *GTOP database* [15] which defines some global optimization spacecraft trajectory problems for interplanetary space missions and lists their best putative solutions, submitted by various international research groups. For the *ESA NPI Day 2010* [46] conference, the performance of MIDACO (at that time in beta version 0.3) was evaluated on this set of test problems.

These benchmark problems are known to be extremely difficult and the submission time between different solutions to these problems range between 6 to 34 months (see Table 6.9). Except the *Messenger Full* problem, MIDACO was applied to every problem with only one test run by default parameters. However, there were no maximum time or evaluation budget assigned to MIDACO, thus it was stopped only, in case the best known solution was reached or no more progress were made upon the current solution (in case of the *Messenger* problems). Table 6.9 reports the solution quality obtained by MIDACO with the corresponding required cpu runtime and the time between the first and last solution submission on the ESA/ACT website [15] for every problem of the test set. Note

that those times can not directly be compared, but the time between the first and last submission at least gives an indication on the difficulty of the problem. The calculations were performed on a PC with a 1.66 GHz clock rate and 1 GB RAM working memory.

Table 6.9: MIDACO 0.3 performance on ESA/ACT GTOP database problems

Problem	n	MIDACO solution	Required Time	Time between first and last submission
Cassini1	6	Best	39 Minutes	6 Month
GTOC1	8	Best* (new)	35 Hours	13 Month
Messenger Full	26	3rd Best*	*separated runs*	34 Month
Messenger	18	4th Best	13 Hours	11 Month
Cassini2	22	Best* (new)	50 Days	14 Month
Rosetta	22	Best	6 Days	6 Month
Sagas	12	Best	12 Hours	*only one submission*

* MIDACO solution is published on the ESA/ACT website [15]

As it can be seen from Table 6.9, MIDACO is able to reach the best known solution in 5 out of 7 cases. In case of the two *Messenger* problems MIDACO did not succeed in reaching the best known solution, but still obtains competitive solutions. The required computation time strongly varies between 39 minutes (Cassini1) and 50 days (Cassini2), which was an exceptional long case were the convergence to the desired solution precision required the vast majority of the cpu time. In case of *GTOC1* and *Cassini2* MIDACO did reveal new best known solutions, which remain the best known ones on the ESA/ACT website [15] since May 2009.

6.5 Interplanetary Space Mission Design

The design of an interplanetary space mission from Earth to Jupiter is considered here. This application is based on the real world mission *Galileo* launched by NASA in Octo-

ber 1989 (see http://solarsystem.nasa.gov/galileo/). The Galileo mission was the first one to implement *gravity assist* maneuvers, where the direction and velocity of the spacecraft is changed due to the gravitational force of a planet. The Galileo probe performed three flybys in total (one at Venus and two at Earth) and several minor flybys at asteroids on its way to Jupiter. Here a general interplanetary space mission model is assumed, considering the flyby planets as discrete optimization variables. The model setup is so general, that it will allow several possible feasible trajectories from Earth to Jupiter, including the Galileo type of mission. Together with the continuous optimization parameters (e.g. for thrusting and flyby altitudes), the mission design forms a challenging MINLP problem. To the best knowledge of the author, this is the first time, that a interplanetary space trajectory optimization problem is considered as MINLP.

6.5.1 Space Mission Layout

The space mission is implemented as an optimal control problem containing several stages, corresponding to different *arcs* of the mission. Here an *arc* describes the time between two major events of the mission, such as flyby or thrusting maneuvers. The mission is characterized by three flyby maneuvers, which is based on the real Galileo mission. For thrusting it assumes one *deep space maneuver* (DSM), which is a thrusting maneuver that happens in space where the influence of any planet can be neglected. Further an escape (from Earth) and a capture (for Jupiter) thrusting maneuver are supposed. Further the flyby altitudes are assumed as continuous optimization variables as well as the time duration of each arc. Figure 6.4 illustrates the mission layout regarding the five arcs and major events.

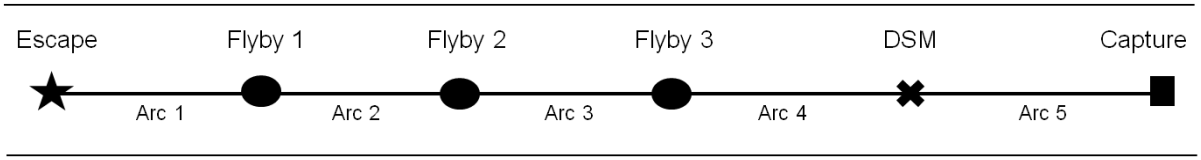

Figure 6.4: MGA-DSM space mission layout regarding arcs and major events

In total, the space mission model consists of 18 continuous optimization variables x and three integer variables y, representing the planet candidates for every flyby in the mission layout (see Fig. 6.4). All nine planets of the solar system are considered as possible flyby candidate. Thus the integer complexity of this MINLP formulation is $9^3 = 729$, whereas only the first four planets of the inner solar system seem to be reasonable flyby candidates. Table 6.10 lists the nine planets with their corresponding integer identification number.

Table 6.10: Planet numeration

Number	Planet
1	Mercury
2	Venus
3	Earth
4	Mars
5	Jupiter
6	Saturn
7	Uranus
8	Neptune
9	Pluto

Table 6.11 lists all optimization variables of the mission with a brief description and assumed lower and upper bounds. Note, that the three thrusting maneuvers are controlled in three cartesian dimensions (X, Y, Z). For gravity assists a minimal and maximal flyby altitude is assumed, which is based on the concrete flyby planet radius. The minimal flyby altitude Alt_{min} is defined as (101)% of the planet radius, where the 1% simulates

the planet atmosphere. Only for the Earth, an atmosphere of 300km (instead of 63.78km, which would be 1% of the Earth radius) is assumed in particular, to take into account satellites orbiting the Earth. The maximal flyby altitude Alt_{max} is defined as $Alt_{max} = 100 \cdot Alt_{min}$, which also allows to model flybys of very little impact on the trajectory. The launch date of the space mission is considered a further continuous optimization variable, where a launch time window of two years (from 1 Jan. 1989 to 31 Dec. 1990) is assumed.

Table 6.11: Optimization variables x (continuous) and y (integer) with bounds

Variable	Description	Lower Bound	Upper Bound
continuous			
x_1	Launch Date	0 (01 Jan. 1989)	730 (31 Dec. 1990)
x_2	Duration of Arc 1	0 (days)	200 (days)
x_3	Duration of Arc 2	0 (days)	400 (days)
x_4	Duration of Arc 3	0 (days)	800 (days)
x_5	Duration of Arc 4	0 (days)	100 (days)
x_6	Duration of Arc 5	0 (days)	1200 (days)
x_7	Thrust Escape (X direction)	-6000.0 (m/sec)	6000.0 (m/sec)
x_8	Thrust Escape (Y direction)	-6000.0 (m/sec)	6000.0 (m/sec)
x_9	Thrust Escape (Z direction)	-3000.0 (m/sec)	3000.0 (m/sec)
x_{10}	Thrust Capture (X direction)	-6000.0 (m/sec)	6000.0 (m/sec)
x_{11}	Thrust Capture (Y direction)	-6000.0 (m/sec)	6000.0 (m/sec)
x_{12}	Thrust Capture (Z direction)	-3000.0 (m/sec)	3000.0 (m/sec)
x_{13}	Thrust DSM (X direction)	-1000.0 (m/sec)	1000.0 (m/sec)
x_{14}	Thrust DSM (Y direction)	-1000.0 (m/sec)	1000.0 (m/sec)
x_{15}	Thrust DSM (Z direction)	-500.0 (m/sec)	500.0 (m/sec)
x_{16}	Altitude Flyby 1	0.00 ($\sim Alt_{min}$)	1.00 ($\sim Alt_{max}$)
x_{17}	Altitude Flyby 2	0.00 ($\sim Alt_{min}$)	1.00 ($\sim Alt_{max}$)
x_{18}	Altitude Flyby 3	0.00 ($\sim Alt_{min}$)	1.00 ($\sim Alt_{max}$)
integer			
y_1	Planet Flyby 1	1 (Mercury)	9 (Pluto)
y_2	Planet Flyby 2	1 (Mercury)	9 (Pluto)
y_3	Planet Flyby 3	1 (Mercury)	9 (Pluto)

The objective function $f(x, y)$ to be minimized is defined as the total ΔV (*change in velocity*) of the mission, which is produced by all the individual thrusting maneuvers.

This is in particular the ΔV_{escape}, $\Delta V_{capture}$ and ΔV_{DSM}. While the ΔV_{DSM} directly corresponds to the thrusting of the DSM, the ΔV_{escape} and $\Delta V_{capture}$ take into account velocity of the escape and capture planet and thus express the velocity change in reference to the planet and not the Sun. The mathematical formulation of the objective function is given in Equation 6.3:

$$
\begin{aligned}
f(x,y) &= \Delta V = \Delta V_{escape} + \Delta V_{capture} + \Delta V_{DSM}, \\
&\text{with} \\
\Delta V_{escape} &= \sqrt{\frac{2\mu^{Earth}}{x_7^2 + x_8^2 + x_9^2}} - \sqrt{2\mu^{Earth}\left(\frac{1}{R_{peregee}^{Earth}} - \frac{1}{R_{apogee}^{Earth} - R_{peregee}^{Earth}}\right)}, \\
\Delta V_{capture} &= \sqrt{\frac{2\mu^{Jupiter}}{x_{10}^2 + x_{11}^2 + x_{12}^2}} - \sqrt{2\mu^{Jupiter}\left(\frac{1}{R_{peregee}^{Jupiter}} - \frac{1}{R_{apogee}^{Jupiter} - R_{peregee}^{Jupiter}}\right)}, \\
\Delta V_{DSM} &= \sqrt{x_{13}^2 + x_{14}^2 + x_{15}^2}.
\end{aligned}
\tag{6.3}
$$

The astrophysical constants used in Equation 6.3 are listed in Table 6.12.

Table 6.12: Gravitation parameter and apsis for Earth and Jupiter

$\mu^{Earth} = 3.986 \cdot 10^{14}$	$R_{peregee}^{Earth} = 6778000$	$R_{apogee}^{Earth} = 42165000$
$\mu^{Jupiter} = 1.267 \cdot 10^{17}$	$R_{peregee}^{Jupiter} = 10^9$	$R_{apogee}^{Jupiter} = 2 \cdot 10^{10}$

The mission is restricted to twelve nonlinear constraints $g_1(x,y), ..., g_{12}(x,y)$ which take into account on the trajectory of the spacecraft. The calculation of the trajectory (nominated as $R^{Spacecraft}(t)$) is not given in detail here, as its implementation takes into account several complex subroutines, which were friendly provided by EADS Astrium (http://www.astrium.eads.net/). Those subroutines model for example the orbit propagation of the spacecraft based on Lagrange coefficients. Here only the mathematical structure of the constraints $g(x,y)$ should be illustrated, which form the MINLP problem.

The full implementation of the Mission (including the subroutines provided by Astrium) can be downloaded at http://www.midaco-solver.com/applications.html for studying or reproduction purposes. Table 6.13 describes the notation used in the constraints for those functions, which are not given in detail here.

Table 6.13: Notation for constraints

Notation	Description
$R^{Spacecraft}(t)$	Position of the spacecraft at time t
$R^{Planet}(t, P)$	Position of planet P at time t
$V^{Planet}(t, P)$	Velocity of planet P at time t
$V_X^{Spacecraft}(t)$	Velocity of the spacecraft (X direction)
$V_Y^{Spacecraft}(t)$	Velocity of the spacecraft (Y direction)
$V_Z^{Spacecraft}(t)$	Velocity of the spacecraft (Z direction)
$rota^{Planet}(P)$	Orbit rotation time (in days) of planet P
$S^{\%}$	Sphere of action radius of a planet (given in percentage $\in [0, 1]$)

The first three constraints model the necessary condition for the flyby maneuvers, that the spacecraft is within the *sphere of action* (roughly speaking the position) of the concrete flyby planet (here y_1, y_2 and y_3) at the time, when the actual flyby maneuver is supposed to happen. This time point depends on the launch date x_1 and the time durations of the arcs $(x_2, ..., x_6)$. The radius of the sphere of action is based on the distance of the concrete flyby planet to the sun. Here the constraints assume, that the spacecraft is within at least $S^{\%}$ percent (with $S^{\%} \in [0, 1]$, where $0 \sim 0\%$ and $1 \sim 100\%$) of this distance, which is considered the *sphere of action*. Equation 6.4 states the mathematical structure of the

first three constraints:

$$
\begin{aligned}
g_1(x,y) =& \|R^{Spacecraft}(x_1+x_2) - R^{Planet}(x_1+x_2,\ y_1)\| \\
&\leqslant S^{\%}\ \|R^{Planet}(x_1+x_2,\ y_1)\|, \\
\\
g_2(x,y) =& \|R^{Spacecraft}(x_1+x_2+x_3) - R^{Planet}(x_1+x_2+x_3,\ y_2)\| \\
&\leqslant S^{\%}\ \|R^{Planet}(x_1+x_2+x_3,\ y_2)\|, \\
\\
g_3(x,y) =& \|R^{Spacecraft}(x_1+x_2+x_3+x_4) - R^{Planet}(x_1+x_2+x_3+x_4,\ y_3)\| \\
&\leqslant S^{\%}\ \|R^{Planet}(x_1+x_2+x_3+x_4,\ y_3)\|.
\end{aligned}
\tag{6.4}
$$

The fourth constraint models the necessary condition, that the spacecraft is within the sphere of action of Jupiter (Planet number 5) at the very end of the mission. This is done analog to the first three constraints. Equation 6.5 states the mathematical structure of the fourth constraint:

$$
\begin{aligned}
g_4(x,y) =& \|R^{Spacecraft}(x_1+x_2+x_3+x_4+x_5+x_6) \\
&- R^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)\| \\
&\leqslant S^{\%}\ \|R^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)\|.
\end{aligned}
\tag{6.5}
$$

The fifth to seventh constraint model the necessary condition, that the velocity of the spacecraft at the very end of the mission is identical to the velocity (and direction) of the target planet, Jupiter. This is to later allow the spacecraft to orbit Jupiter. As the total velocity of Jupiter itself ($\|V^{Planet}(t,5)\|$) is undirected, three constraints are necessary to take into account the specific directions (in X,Y, and Z). The same tolerance $S^{\%}$ for the *sphere of action* (described above) is applied here to the total velocity of the target planet, to allow some tolerance in fulfilling these constraints. Equation 6.6 states the

mathematical structure of the fifth to seventh constraint:

$$\begin{aligned} g_5(x,y) =& |V_X^{Spacecraft}(x_1+x_2+x_3+x_4+x_5+x_6) - \\ & V_X^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)| \\ & \leqslant S^{\%}\ \|V^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)\|, \end{aligned}$$

$$\begin{aligned} g_6(x,y) =& |V_Y^{Spacecraft}(x_1+x_2+x_3+x_4+x_5+x_6) - \\ & V_Y^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)| \\ & \leqslant S^{\%}\ \|V^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)\|, \end{aligned} \tag{6.6}$$

$$\begin{aligned} g_7(x,y) =& |V_Z^{Spacecraft}(x_1+x_2+x_3+x_4+x_5+x_6) - \\ & V_Z^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)| \\ & \leqslant S^{\%}\ \|V^{Planet}(x_1+x_2+x_3+x_4+x_5+x_6,\ 5)\|. \end{aligned}$$

Some additional constraints are imposed, to avoid feasible, but undesired solutions. Those undesired solutions might attract the optimization algorithm, but are not of any relevance and should therefore be avoided. The eighth constraint impose, that not all flyby maneuvers happen at the planet Earth. Equation 6.7 states the mathematical structure of the eighth constraint:

$$g_8(x,y) = \begin{cases} \infty & \text{, if } y_1 = y_2 = y_3, \\ 0 & \text{, else.} \end{cases} \tag{6.7}$$

The ninth to twelfth constraint impose, that if two successive flybys happen at the same planet, the time duration between those flybys must be at least greater or equal than half of the total rotation time of the flyby planet (in respect to the sun). Equation 6.8 states

the mathematical structure of the ninth to twelfth constraint constraint:

$$
\begin{aligned}
g_9(x,y) &= \begin{cases} \frac{rota^{Planet}(y_1)}{2} - x_2 \leqslant 0 & \text{, if } y_1 = 3, \\ 0 & \text{, else,} \end{cases} \\
g_{10}(x,y) &= \begin{cases} \frac{rota^{Planet}(y_2)}{2} - x_3 \leqslant 0 & \text{, if } y_2 = y_1, \\ 0 & \text{, else,} \end{cases} \\
g_{11}(x,y) &= \begin{cases} \frac{rota^{Planet}(y_3)}{2} - x_4 \leqslant 0 & \text{, if } y_3 = y_2, \\ 0 & \text{, else,} \end{cases} \\
g_{12}(x,y) &= \begin{cases} \frac{rota^{Planet}(5)}{2} - (x_5 + x_6) \leqslant 0 & \text{, if } 5 = y_3, \\ 0 & \text{, else.} \end{cases}
\end{aligned}
\tag{6.8}
$$

6.5.2 Numerical Results

MIDACO has been used to solve this application in a two step approach. In a first step, a number of several test runs using different random seeds are performed on the full mission model using a moderate parameter of $S^{\%} = 0.03$ (which is 3%) for the accuracy of the sphere of action around the planets. As the accuracy of the sphere of action is a crucial parameter in the model, this moderate accuracy will allow MIDACO to identify possible feasible mission trajectories more easily, while concentrating more on the integer complexity of the problem. In a second step, the most promising solutions found within the first step should then be further refined by MIDACO, assuming a higher accuracy of 0.5% for the sphere of action.

Table 6.15 lists the results of the first optimization step, where MIDACO is applied 10

times on the space mission model using different random seeds and a moderate precision of 3% for the sphere of action in the model. Every test run was performed for a duration of one hour on a PC with an Intel Xeon CPU E5640 (2.67GHz clock rate, 4GB RAM) which corresponds to some hundred million function evaluation. Note, that such a high amount of function evaluation is nothing unusual for a stochastic algorithm applied on a difficult problem. As the MIDACO software is capable of processing millions of iterates in seconds, the total cpu runtime of one hour remains here still reasonable.

Table 6.14: Abbreviations for Table 6.15

Abbreviation	Explanation
Run	Number of test run
Launch	Date for mission departure from Earth
ΔV	Objective function value (m/sec)
Duration	Duration of total mission (Years)
FlyBy 1	Planet selected by MIDACO for 1st gravity assist maneuver
FlyBy 2	Planet selected by MIDACO for 2nd gravity assist maneuver
FlyBy 3	Planet selected by MIDACO for 3rd gravity assist maneuver

Table 6.14 explains the abbreviations used in Table 6.15.

Table 6.15: 10 test runs by MIDACO on mission model with 3% sphere of action

Run	Launch	ΔV	Duration	FlyBy 1	FlyBy 2	FlyBy 3
1	6 Nov. 1989	2553	5.88	Venus	Earth	Earth
2	30 Nov. 1989	3310	4.83	Venus	Earth	Earth
3	30 Nov. 1989	3218	4.88	Venus	Earth	Earth
4	10 Jul. 1989	3390	3.97	Venus	Earth	Mars
5	20 Nov. 1989	2890	4.77	Venus	Earth	Earth
6	23 May 1989	2759	5.35	Earth	Venus	Earth
7	21 Mar 1989	*infeasible*	4.85	Earth	Earth	Mars
8	13 Apr. 1989	3290	4.54	Earth	Venus	Earth
9	30 Nov. 1989	3289	4.79	Venus	Earth	Earth
10	16 Sep. 1989	2684	6.10	Venus	Earth	Earth

As it can be seen from Table 6.15, MIDACO did reveal a number of possible mission trajectory, based on different flyby planet candidates. Those missions vary strongly in their characteristics, as it can be seen from the differences in the launch date, objective function values ΔV and total flight durations. Only in one case (test run number 7), MIDACO did not succeed to find a feasible mission trajectory. The integer combination mostly attracted by MIDACO is (Venus, Earth, Earth), which is indeed the same combination as used in the original Galileo mission. Besides this combination, the combination (Venus, Earth, Mars) found in test run number 4 seems somewhat interesting. This mission has the worst (in esp. highest) objective function value, but also the shortest flight duration.

In a second optimization step, the solutions corresponding to test run number 1 (named Mission1) and number 4 (named Mission4)from Table 6.15 should now be refined by MIDACO, assuming a more precise accuracy of 0.5% for the sphere of action around the planets. For this purpose, those solutions are given to MIDACO as starting point and the *QSTART* parameter explained in Section 4.2 is activated (using a value of 10000) for a more efficient search in the vicinity of the submitted initial solution. MIDACO is applied to the refined model for one hour again for both initial solutions (in esp. Mission1 and Mission4 from Table 6.15). Table 6.16 shows the solutions of the refinement of Mission1 and Mission4 with details on the individual flyby maneuvers. Table 6.16 also compares these two missions generated by MIDACO with the original Galileo mission regarding their main characteristics.

Table 6.16: Comparison between original Galileo and MIDACO Missions

	Galileo Mission	Mission1 refine 0.5 %	Mission4 refine 0.5 %
Launch	18 Oct. 1989	8 Nov. 1989	6 Jul. 1989
Duration	6.14 Years	6.14 Years	4.15 Years
ΔV	*unknown*	3,350 m/sec	5,177 m/sec
1st Flyby			
Planet	Venus	Venus	Venus
Date	10 Feb. 1990	23 Feb. 1990	21 Jan. 1990
Altitude	16,000km	28,901km	3,013km
2nd Flyby			
Planet	Earth	Earth	Earth
Date	8 Dec. 1990	5 Dec. 1990	4 Sep. 1990
Altitude	960km	473,191km	1,754km
3rd Flyby			
Planet	Earth	Earth	Mars
Date	8 Dec. 1992	4 Dec. 1992	31 Dec. 1990
Altitude	303km	300km	39km

Analyzing the Missions from Table 6.16, the structural difference between Mission1 and Mission4 is evident. Mission4 is however in so far interesting in respect to Mission1, as it provides a much shorter ($\sim$ 32.4%) flight duration to the price of an equivalent increased ($\sim$ 35.3%) objective function value. More interestingly is the striking similarity between Mission1 and the original Galileo mission. With a shift of about 22 days regarding the launch date, these two mission share exactly the same flight duration and are closely related regarding all the characteristics of their gravity assist maneuvers, except the flyby altitude at the 2nd flyby. Here a significant difference between 960km (Galileo) and 473,191 km (Mission1) occurs. However, this difference can be well explained by taking into account, that for the original Galileo mission the observation of asteroids were an objective, while asteroids were not considered in the model formulation here. Both missions perform their first Earth flyby in December 1990. The Galileo mission performs its first flyby at Earth at a moderate altitude of 960km in order to significantly increase

the semi major axis of its orbit to visit the Gaspra asteroid in October 1991. This new orbit is then rotated one time by the Galileo probe, before it performs its second flyby at Earth two years later in 1992. As in this model here asteroids were not considered, this maneuver is not a target in Mission1. Therefore Mission1 performs its first flyby at Earth at a very large altitude of 473,191 km in order to have only a very small gravitational impact on its trajectory. This way Mission1 can perform two rotations of its near Earth orbit in a row, before it performs its second Earth flyby two years later in December 1992 like the original Galileo mission.

Figure 6.5 shows the space mission trajectory of the original Galileo mission (image taken from *Wikimedia* (http://commons.wikimedia.org)) and the Mission1 generated by MIDACO. The coincidence of the major events of both missions can be well observed. Also the difference in the trajectories between the 2nd and 3rd flyby can be seen: While Galileo performs one orbit rotation visiting Gaspra, Mission1 remains in a close Earth orbit rotating it two times.

Note, that **Video Animations** of the MIDACO Mission1 and Mission4 are available at http://www.midaco-solver.com/applications.html for downloading.

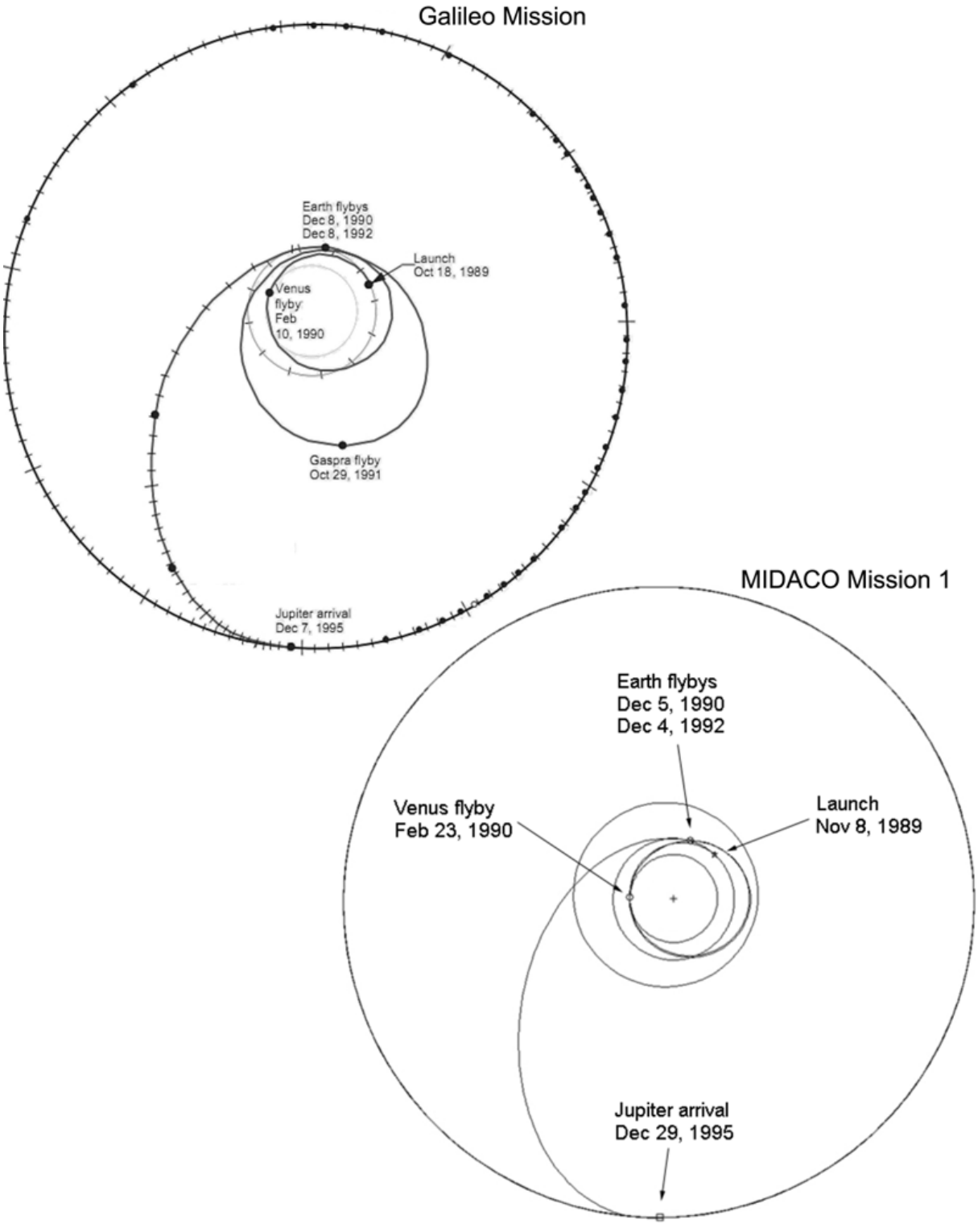

Figure 6.5: Space trajectories of the NASA Galileo mission and MIDACO Mission1

6.5.3 Space Mission Design: Conclusions

For the first time a multi gravity assist interplanetary space mission was considered in an MINLP formulation, considering flyby planet candidates as discrete decision variables. As space trajectory optimization problems are known to be very difficult (see Section 6.4), this MINLP approach can be considered as exceptionally challenging. In accordance to the Galileo mission by NASA in 1989, a general space mission model for transfers from Earth to Jupiter was formulated, based on three flyby maneuvers. In a two step optimization process, it could be shown, that MIDACO is able to generate feasible space trajectories in a reasonable time and accuracy fully automatically. The results from Table 6.15 indicate, that the space mission model setting was general enough, to allow several feasible space trajectories (which implies a sufficient large search space). Among some different space mission trajectory candidates, MIDACO did reveal the Galileo type of mission with high probability (7 out of 10 cases). The accuracy of the sphere of action around the planets was identified as crucial model parameter. Within a second optimization step, MIDACO was able to refine its generated missions to a sufficient accuracy of 0.5% and the fully automatically generated mission by MIDACO showed an intriguing coincidence with the characteristics of the original Galileo mission from 1989.

6.6 Multiple-Stage Launch Vehicle Ascent Problem

The ascent of a multiple-stage launch vehicle is considered here. The model is based on a Delta III rocket (*The Boeing Company*) and was originally introduced by Benson [5]. In its original (continuous) formulation, the model is used as a benchmark in the open literature Huntington [31] and in well known optimal control software packages like

GPOPS [37].

Here, the original model as given in GPOPS [37] is extended by additional mixed integer decision variables and nonlinear constraints, creating a challenging mixed integer multi stage optimal control problem. While in the original formulation the type and number of strap on boosters used for the rocket propulsion is fixed, those two engineering design aspects are formulated as discrete decision variables here. It can be shown, that by introducing those additional degrees of freedom, significant improvements in the objective function can be gained in comparison to the original model. Additionally incorporated constraints on the maximal dynamic pressure for the vehicle and a virtual financial budget (based on the type of strap on boosters employed) ensure, that the feasible solutions are still reasonable.

The objective in this application is to maximize the remaining fuel of the vehicle while maneuvering it from the ground to a low earth target orbit. In the following, the model formulation in GPOPS [37] is closely followed while the above mentioned extensions are especially highlighted in an individual subsection.

Note that the implementation of this application discussed in the following can be downloaded at `http://www.midaco-solver.com/applications.html`.

6.6.1 Vehicle Properties

The launch vehicle consists of two main stages and contains nine strap-on solid rocket boosters. The flight of the vehicle to its target orbit can be divided into four different phases. The first flight phase considers the vehicle on the ground at time t_0. The main engine and a number of boosters ignite (the concrete number of boosters is considered

an optimization variable here). At time t_1 the number of boosters ignited at t_0 are depleted and the remaining dry mass is jettisoned. In the following second flight phase, the remaining strap-on boosters ignite and are depleted at time t_2. The third flight phase does only consider propulsion by the main engine of the vehicle stage 1. The fourth flight phase begins when the main engine fuel is finished and the dry mass associated with the main engine is ejected at time t_3. During flight phase four only the main engine of the vehicle stage 2 is used for propulsion. The flight phase four ends at time t_4, when the vehicle reaches the desired low earth target orbit. Note that the solid boosters and main engine burn for their entire duration (meaning t_1, t_2, and t_3 are fixed), while the second stage engine is shut off when the target orbit is achieved, therefore t_4 is an optimization variable. The mass and propulsion properties of the two vehicle stages are taken from GPOPS [37] and listed in Table 6.17.

Table 6.17: Vehicle mass and propulsion properties

	Stage 1	Stage 2
Total Mass (kg)	104380	19300
Propellant Mass (kg)	95550	16820
Engine Thrust (N)	1083100	110094
Specific Impulse (sec)	301.7	467.2
Number of Engines	1	1
Burn Time (sec)	7261	700

Dynamic Model

The equations given in 6.9 express the Cartesian coordinates (earth-centered) of a non-

lifting mass point in flight over a spherical rotating planet

$$\begin{aligned} \dot{r} &= v, \\ \dot{v} &= -\frac{\mu}{\|r\|^3} + \frac{T}{m}u + \frac{D}{m}, \\ \dot{m} &= -\frac{T}{g_0 I_{sp}}, \end{aligned} \tag{6.9}$$

where $r = (x, y, z)'$ is the (earth-centered) Cartesian position of the mass point, $r = (v_x, v_y, v_z)'$ is the velocity, μ is the gravitational parameter, T is the vacuum thrust, m is the mass, g_0 is the acceleration due to gravity at sea level, I_{sp} the specific impulse of the engine, $u = (u_x, u_y, u_z)'$ is the thrust direction and $D = (D_x, D_y, D_z)'$ is the drag force. The drag force is defined as

$$D = -\frac{1}{2}\rho S C_D \|v_{rel}\| v_{rel} \ , \tag{6.10}$$

where C_D is the drag coefficient, S is the reference area, ρ is the atmospheric density and v_{rel} is the earth relative velocity, where v_{rel} is given as

$$v_{rel} = v - \Omega \times r, \tag{6.11}$$

where Ω is the angular velocity of the earth relative to the inertial reference frame. The atmospheric density is modelled as the exponential function

$$\rho = \rho_0 e^{\frac{-h}{H}}, \tag{6.12}$$

where ρ_0 is the atmospheric density at sea level, $h = \|r\| - R_e$ is the altitude, R_e is the equatorial radius of the earth and H is the density scale height. Table 6.18 contains the numerical values for the constants used in the vehicle model.

Table 6.18: Constants used in the launch vehicle model

Constant	Value
Payload mass (kg)	4164
S (m^2)	$4\ \pi$
C_D	0.5
ρ_0 (kg/m^3)	1.225
H (m)	7200
t_1 (sec)	75.2
t_2 (sec)	150.4
t_3 (sec)	261
R_e (m)	6378145
Ω (rad/s)	$7.29211585 \times 10^{-5}$

The launch vehicle starts on the ground at rest (relative to the earth) at time t_0, so that the earth centered initial conditions are

$$\begin{aligned} r(t_0) &:= r_0 = (R_e cos\Phi_0, 0, R_e sin\Phi_0)', \\ v(t_0) &:= v_0 = \Omega \times r_0, \\ m(t_0) &:= m_0 = 301454(kg), \end{aligned} \tag{6.13}$$

where $\Phi_0 = 28.5^\circ$ and corresponds to the geocentric latitude of Cape Canaveral (Florida) and it is arbitrarily assumed that the inertially fixed axes are such that the initial inertial longitude is zero. The terminal constraints define the target geosynchronous transfer orbit, which is defined in terms of orbital elements as

$$\begin{aligned} a_f &= 24361.14 \text{ km}, \\ e_f &= 0.7308, \\ i_f &= 28.5^\circ, \\ \Omega_f &= 269.8^\circ, \\ \omega_f &= 130.5^\circ, \end{aligned}$$

where a is the semimajor axis, e is the eccentricity, i is the inclination, Ω is the inertial

longitude of the ascending node and ω is the argument of perigee. The true anomaly v is considered free, as no location in the terminal orbit is specified as constraint. Besides the primary constraint of reaching the terminal orbit, a state path constraint keeps the altitude of the vehicle above the earth surface and is given as

$$|r| \geqslant R_e, \tag{6.14}$$

where R_e is the equatorial radius of the earth. In contrast to the original model formulation in GPOPS [37], no equality path constraint of the form

$$\|u\|^2 = u_x^2 + u_y^2 + u_z^2 = 1 \tag{6.15}$$

is necessary here, because the Pitch and Yaw system is applied to control the vehicle that inherently satisfies the above equality path constraint. The transformation from Pitch (Ψ) and Yaw (Φ) to the cartesian thrust direction $u = (u_x, u_y, u_z)'$ is given by

$$\begin{aligned} u_x &= cos(\Psi)cos(\Theta), \\ u_y &= cos(\Psi)sin(\Theta), \\ u_z &= sin(\Psi), \end{aligned} \tag{6.16}$$

where $\Psi \in [-\pi, \pi]$ and $\Theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. The model further contains linkage constraints between the different phases regarding the position r, the velocity v and the mass m of the vehicle. Those linkage constraints are active at the end of phases 1,2 and 3 and the

start of phases 2, 3 and 4, respectively as

$$\begin{aligned} r^{(p)}(t_f) - r^{(p+1)}(t_0) &= 0, \\ v^{(p)}(t_f) - v^{(p+1)}(t_0) &= 0, \\ m^{(p)}(t_f) - m^{(p)}_{dry} - m^{(p+1)}(t_0) &= 0, \end{aligned} \tag{6.17}$$

where the subscript (p) denotes the phase number (in esp. $p = \{1, 2, 3\}$) and t_0 and t_f denote the start and final time points of the corresponding phase. The linkage constraints force the position r and velocity v to be continuous regarding the phase transition. The linkage constraint on the mass $m^{(p)}$ at each of the phase interfaces corresponds with an instantaneous drop of the dry mass $m^{(p)}_{dry}$ of the particular phase (p). Therefore the mass trajectory is not continuous at the stage interfaces, when mass is ejected. Here, mass drops at the ends of phases 1,2 and 3, when the dry mass of the strap-on boosters or the first main stage is depicted. In contrast to the original formulation of the model in GPOPS [37], the amount of dry mass associated with the strap-on boosters dropped at the ends of phases 1 and 2 depends on the number of boosters chosen, which is an integer decision variable (see Subsection 6.6.2).

The objective of the problem is to find an optimal control (and corresponding trajectory) that maximizes the remaining mass of the vehicle at the end of phase 4. This objective is expressed as minimization of the cost functional J given as

$$J = -m^{(4)}(t_f), \tag{6.18}$$

subject to the above defined conditions and constraints.

6.6.2 Mixed Integer Extensions

The launch vehicle model is extended by some discrete decision variables regarding the number and type of strap-on boosters used. While in the original formulation in GPOPS [37] the number of strap-on boosters is considered fixed (6 booster for phase 1 and 3 booster for phase 2), here the number of strap-on booster applied in the first phase is a decision variable and denoted by $B_1 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The number B_2 of strap-on booster applied in the second phase is then recursively computed by $B_2 = 9 - B_1$ and therefore not considered a decision variable. Further to the number of boosters applied, also the type of booster is considered to be variable here. For simplicity and practical relevance the selection of booster types is restricted to phase 1 and 2. Hence, all boosters of phase 1 or 2 are assumed to be of the same type. Based on the type of strap-on booster defined in the original formulation, four new types are generated. Those four new types do vary in 10 and 25 percentage respectively to the properties of the original booster defined in GPOPS [37]. Table 6.19 lists all five possible strap-on booster types with their properties. Note, that in Table 6.19 the booster type 3 corresponds to the original booster considered in GPOPS [37].

Table 6.19: Abbreviations for Table

Type	Thrust Power (N)	Mass $_{Total}$ (kg)	Mass $_{Propellant}$ (kg)	Cost
1	471375	14468	12758	0.75
2	565650	17361	15309	0.90
3	628500	19290	17010	1.00
4	691350	21219	18711	1.10
5	785625	24113	21263	1.25

Table 6.19 also introduces a new property for the five considered booster types. This is a virtual cost which is based on the original strap-on booster. Hence the cost for booster

type 3 is defined as 1.00 while the costs for all other boosters correspond with their 10 or 25 percentage property in- or decrease. The here introduced cost property of the booster types is later used to define some financial constraint that restricts the feasible choice of boosters.

6.6.3 Additional Constraints

Additional constraints are considered here. This is a maximal dynamic pressure constraint which ensures, that the dynamic pressure on the launch vehicle never exceeds 50.000 $\mathrm{N/m^2}$. This constraint is to ensure that the choice of boosters implies a reasonable dynamic pressure behavior that does not damage the space vehicle. The dynamic pressure constraint is given by

$$\frac{1}{2}P||v||^2 \leqslant 50000, \tag{6.19}$$

where P is the atmospheric pressure calculated as

$$P = 1.249512\ e^{-\frac{||r||}{6900}}. \tag{6.20}$$

A further constraint is introduced as maximal financial budget regarding the type of strap-on boosters. The maximal available budget is considered here as 9, which is based on the original model formulation applying 9 boosters of type 3 (which has cost 1 as listed in Table 6.19). The financial budget constraint is formulated as

$$B_1T_1 + B_2T_2 \leqslant 9, \tag{6.21}$$

where $B_2 = 9 - B_1$ is the number of strap-on boosters used in the second phase. Last, a vertical take-off constraint is imposed which ensures, that the vehicle launches vertical during the first five seconds of its flight. This constraint is considered a security procedure that increases the realistic relevance of the launch vehicle start. The vertical take-off constraint is imposed as a fixed control $\overline{u} = (\overline{u}_x, \overline{u}_y, \overline{u}_z)'$ during the first five seconds of phase 1, where $\overline{u}$ is given by

$$\overline{u} = \frac{r}{||r||}. \tag{6.22}$$

6.6.4 Numerical Results

To solve the above described (mixed integer multi stage) optimal control problem, an reduced direct approach is followed, where a Runge Kutta method of order 2 is applied to integrate the ordinary differential equations 6.9. A discretization of 15 grid points is applied in each of the four flight phases, which is identical to the number of grid points used in GPOPS [37]. The resulting MINLP has then 128 decision variables, where 3 of them are discrete (this is B_1, T_1, T_2; see Table 6.20), and 127 constraints, where 5 of them are equality constraints defining the target orbit. The integer complexity of the MINLP is 250 ($10 \times 5 \times 5$).

A hybrid optimization strategy is implemented. This strategy first applies MIDACO to search the MINLP formulation on the full mixed integer search domain for a fixed time budget. In a second step, an SQP algorithm (SQP-Filtertoolbox by Prof. Gerdts) is applied, using the solution given by MIDACO as initial starting point. For the SQP method the 3 discrete decision variables are fixed to the values given by the MIDACO solution, hence the SQP is applied only on the continuous search domain of the problem

and considered a refinement technique for the MIDACO solution. More details on the hybridization of MIDACO and SQP can be found in Section 4.3.

Before optimizing the full MINLP an investigation on the impact of the discrete decision variables is performed. Table 6.21 presents the best known results for the launch vehicle problem in respect to different combinations of discrete variables that have been fixed. Only feasible combinations with $B_1 \geqslant 6$ are considered in Table 6.21. For integer combinations with $B_1 = 9$ results are only reported in respect to T_1 because the influence of T_2 is literally zero in this scenario ($B_2 = 9 - B_1$). Table 6.20 contains the abbreviations for Table 6.21. Note, that in Table 6.21 the integer combination $\{B_1 = 6, T_1 = 3, T_2 = 3\}$ expresses the original booster configuration as assumed in GPOPS [37]. While in GPOPS [37] an optimal objective function of 7529.71kg is reported, here a value of 7504.48kg is given to the corresponding integer combination $\{6, 3, 3\}$. This difference find its explanation in the additional constraints and different integration approach applied here.

From the results of Table 6.21 it can be seen, that the discrete booster decision variables do have a significant impact on the objective function. In esp. seven different combinations can be identified, that improve the result in respect to the original combination of $\{6, 3, 3\}$. The best known result in Table 6.21 corresponds to the integer combination of $\{8, 3, 3\}$. The non-intuitive and diverse impact of the different integer combination on the solution exemplifies the complexity of this MINLP formulation very well.

Table 6.20: Abbreviations for Table 6.21

Abbreviation	Explanation
Booster-Config.	Booster Configuration: B_1, T_1, T_2
B_1	Number of active booster in first phase
T_1	Type of booster used in first phase
T_2	Type of booster used in second phase
Best known $f(x, y)$	Best known objective function for corresponding booster configuration

Table 6.21: Enumeration over all (feasible) booster configurations with $B_1 \geqslant 6$

Booster-Config.				Booster-Config.			
B_1	T_1	T_2	Best known $f(x,y)$	B_1	T_1	T_2	Best known $f(x,y)$
6	1	1	-6685.71	8	1	1	-6848.21
6	1	2	-6808.53	8	1	2	-6900.99
6	1	3	-6884.45	8	1	3	-6935.36
6	1	4	-6955.92	8	1	4	-6969.11
6	1	5	-7055.32	8	1	5	-7018.56
6	2	1	-7075.93	8	2	1	-7297.53
6	2	2	-7195.10	8	2	2	-7228.32
6	2	3	-7269.14	8	2	3	-7381.77
6	2	4	-7339.15	8	2	4	-7414.42
6	3	1	-7315.13	8	2	5	-7321.22
6	3	2	-7431.81	8	3	1	**-7565.08**
6	*3*	*3*	*-7504.48*	8	3	2	**-7614.97**
6	4	1	**-7539.17**	8	3	3	**-7647.50**
7	1	1	-6789.90	9	1	-	-6855.30
7	1	2	-6883.25	9	2	-	-7324.23
7	1	3	-6942.60	9	3	-	**-7599.88**
7	1	4	-6999.74				
7	1	5	-7081.49				
7	2	1	-7213.85				
7	2	2	-7303.58				
7	2	3	-7360.66				
7	2	4	-7415.64				
7	2	5	-7494.50				
7	3	1	-7271.36				
7	3	2	**-7556.82**				
7	3	3	**-7612.70**				

Table 6.22 and Table 6.22 present the results of 30 individual test runs on the multiple-stage launch vehicle problem using a fixed budget of 600 (10 Minutes) respectively 7200 (2 Hours) seconds for MIDACO. For every run, a different random seed is used for MIDACO. The different integer combinations and objective function values to the solutions found by MIDACO are reported along the number of function evaluation and cpu-time needed

by MIDACO. The solutions by the SQP method, which uses the MIDACO solutions as starting points, are reported together with the number of function evaluation (including those for gradient approximations) and cpu times. The SQP algorithm were called with a maximum number of 1000 iterations. Note that the SQP may stop earlier, if its convergence criteria is satisfied, while MIDACO always performs over its defined cpu-time budget.

Table 6.22: 30 runs by MIDACO (max time = 600) + SQP (max iter=1000)

	Booster-Config.			SQP			MIDACO		
Run	B_1	T_1	T_2	$f(x,y)$	Eval	Time	$f(x,y)$	Eval	Time
1	8	3	3	-7647.50	327060	322.6	-6857.99	222866	600.0
2	9	3	1	-7599.88	274208	292.9	-7163.86	239657	600.0
3	9	3	5	-7599.88	280700	376.8	-7065.85	235807	600.0
4	9	3	5	-7599.88	271308	259.6	-6972.28	198351	600.0
5	8	3	3	-7647.50	306974	282.5	-6963.09	281567	600.0
6	8	3	3	-7647.50	270452	258.3	-6920.35	269536	600.0
7	9	3	5	-7599.88	269558	262.1	-7038.81	262407	600.0
8	8	3	3	-7647.50	281106	264.3	-6820.98	270953	600.0
9	7	3	3	-7612.85	350416	373.0	-6956.02	261879	600.0
10	8	3	3	-7647.50	145076	139.4	-6996.72	268385	600.0
11	8	2	5	-7462.25	330614	314.2	-6918.20	266153	600.0
12	8	3	3	-7647.50	352408	336.5	-6889.87	266954	600.0
13	9	3	5	-7599.88	297954	377.8	-6972.39	258888	600.0
14	8	3	3	-7647.50	339916	368.0	-6818.03	208452	600.0
15	9	3	2	-7599.88	350742	321.5	-7024.97	272402	600.0
16	8	3	3	-7647.47	359535	334.2	-6914.47	270246	600.0
17	7	3	3	-7612.85	362934	367.2	-6977.27	239453	600.0
18	9	3	1	-7599.88	310868	315.7	-6954.59	258475	600.0
19	9	3	5	-7599.87	363478	369.6	-7036.21	245399	600.0
20	9	3	2	-7599.88	343712	351.9	-6797.39	257383	600.0
21	8	3	3	-7647.50	334478	337.0	-6971.25	252131	600.0
22	8	3	3	-7647.50	306314	311.1	-6852.83	254798	600.0
23	8	3	3	-7647.50	305118	299.1	-6929.07	250874	600.0
24	9	3	3	-7599.88	280658	281.6	-7031.31	254307	600.0
25	8	3	3	-7647.50	270780	252.8	-6911.24	212809	600.0
26	6	4	1	-7539.17	364376	335.3	-6814.24	267337	600.0
27	9	3	5	-7599.88	323402	297.5	-6834.61	273553	600.0
28	6	4	1	-7539.17	356594	315.2	-6872.51	287757	600.0
29	9	3	3	-7599.88	288322	265.3	-6876.98	278023	600.0
30	8	3	3	-7647.50	367318	429.2	-7080.80	278160	600.0
Average over all runs:				-7612.74	312879	313.7	-6941.14	255498	600.0

Table 6.23: 30 runs by MIDACO (max time = 7200) + SQP (max iter=1000)

	Booster-Config.			SQP			MIDACO		
Run	B_1	T_1	T_2	$f(x,y)$	Eval	Time	$f(x,y)$	Eval	Time
1	9	3	1	-7599.88	357908	317.7	-7419.65	3455790	7200.0
2	9	3	1	-7599.88	353114	315.1	-7449.22	3450447	7200.0
3	8	3	3	-7647.50	363366	321.1	-7502.77	3443609	7200.0
4	9	3	1	-7599.88	267022	236.8	-7419.91	3449060	7200.0
5	9	3	5	-7599.88	309848	274.6	-7418.63	3460976	7200.0
6	9	3	1	-7599.88	173384	153.8	-7436.00	3472466	7200.0
7	9	3	1	-7599.88	346444	307.3	-7555.53	3456612	7200.0
8	9	3	4	-7599.88	265638	234.8	-7369.10	3457577	7200.0
9	7	3	3	-7567.75	6713	6.4	-7565.33	3445493	7200.0
10	9	3	4	-7599.88	284148	254.1	-7524.85	3445318	7200.0
11	8	3	3	-7524.57	7379	7.1	-7519.89	3447985	7200.0
12	8	3	3	-7647.50	354988	313.6	-7481.90	3459946	7200.0
13	9	3	1	-7599.88	270324	240.1	-7444.49	3453002	7200.0
14	8	3	3	-7647.50	363938	322.9	-7479.16	3451839	7200.0
15	9	3	5	-7599.88	266138	235.9	-7500.50	3464034	7200.0
16	9	3	5	-7599.88	301198	266.8	-7519.93	3481049	7200.0
17	9	3	3	-7599.88	344342	307.8	-7456.35	3450507	7200.0
18	9	3	1	-7599.88	273766	242.3	-7528.82	3454741	7200.0
19	9	3	1	-7599.88	298972	267.0	-7527.04	3458943	7200.0
20	9	3	4	-7599.88	324916	290.1	-7431.08	3468826	7200.0
21	9	3	5	-7599.88	355510	317.4	-7498.23	3487475	7200.0
22	9	3	5	-7599.88	341446	322.4	-7430.29	3042720	7200.0
23	8	3	3	-7647.43	309588	370.1	-7536.62	2879186	7200.0
24	9	3	5	-7599.88	349166	425.9	-7460.48	2435917	7200.0
25	8	3	3	-7513.12	7360	9.2	-7505.67	2568537	7200.0
26	6	4	1	-7539.17	361342	332.9	-7434.57	2871096	7200.0
27	9	3	5	-7599.88	313390	313.4	-7348.84	3060132	7200.0
28	9	3	3	-7599.88	263188	347.4	-7475.90	3150988	7200.0
29	8	3	3	-7647.50	355292	321.1	-7470.97	2873982	7200.0
30	8	3	3	-7647.50	365252	327.5	-7491.21	3336049	7200.0
Average over all runs:				-7600.91	285169	266.8	-7473.43	3294476	7200.0

In case of Table 6.22 MIDACO reveals the best known integer combination of $\{8, 3, 3\}$ in 13 out of 30 cases (~ 43 %). In 12 cases the integer combination of $\{9, 3, -\}$ has been found.

Only one time (Run 11) an integer combination is found by MIDACO, that corresponds to a solution that is less optimal than the original combination of $\{6, 3, 3\}$. The average objective function value of the MIDACO solutions is 6941.14 kg corresponding to an average of 255498 function evaluation. The SQP algorithm is able to successfully refine all MIDACO solutions to the best known solutions presented in Table 6.21, requiring 312879 function evaluation and about 5 Minutes on average.

In case of Table 6.23 MIDACO reveals the best known integer combination of $\{8, 3, 3\}$ in 8 out of 30 cases ($\sim$ 27 %). In 20 cases the integer combination of $\{9, 3, -\}$ has been found. In all cases MIDACO reveals integer combination, that corresponds to solutions that are more attractive than the original combination of $\{6, 3, 3\}$. The average objective function value of the MIDACO solutions is 7473.43 kg corresponding to an average of 3294476 function evaluation. The SQP algorithm is able to successfully refine the MIDACO solutions to the best known solutions presented in Table 6.21 in 27 out of 30 cases. In three cases (Run 9, Run 11 and Run 25) the SQP algorithm stops prematurely.

Figure 6.6 contains plots regarding the best known solution to the multiple-stage launch vehicle problem corresponding to the integer combination $\{8, 3, 3\}$. This is in particular the altitude, control, velocity, mass, energy transfer and dynamic pressure progression of the launch vehicle during its total flight. It can be seen, that the behavior is very similar to the original solution reported in GPOPS [37].

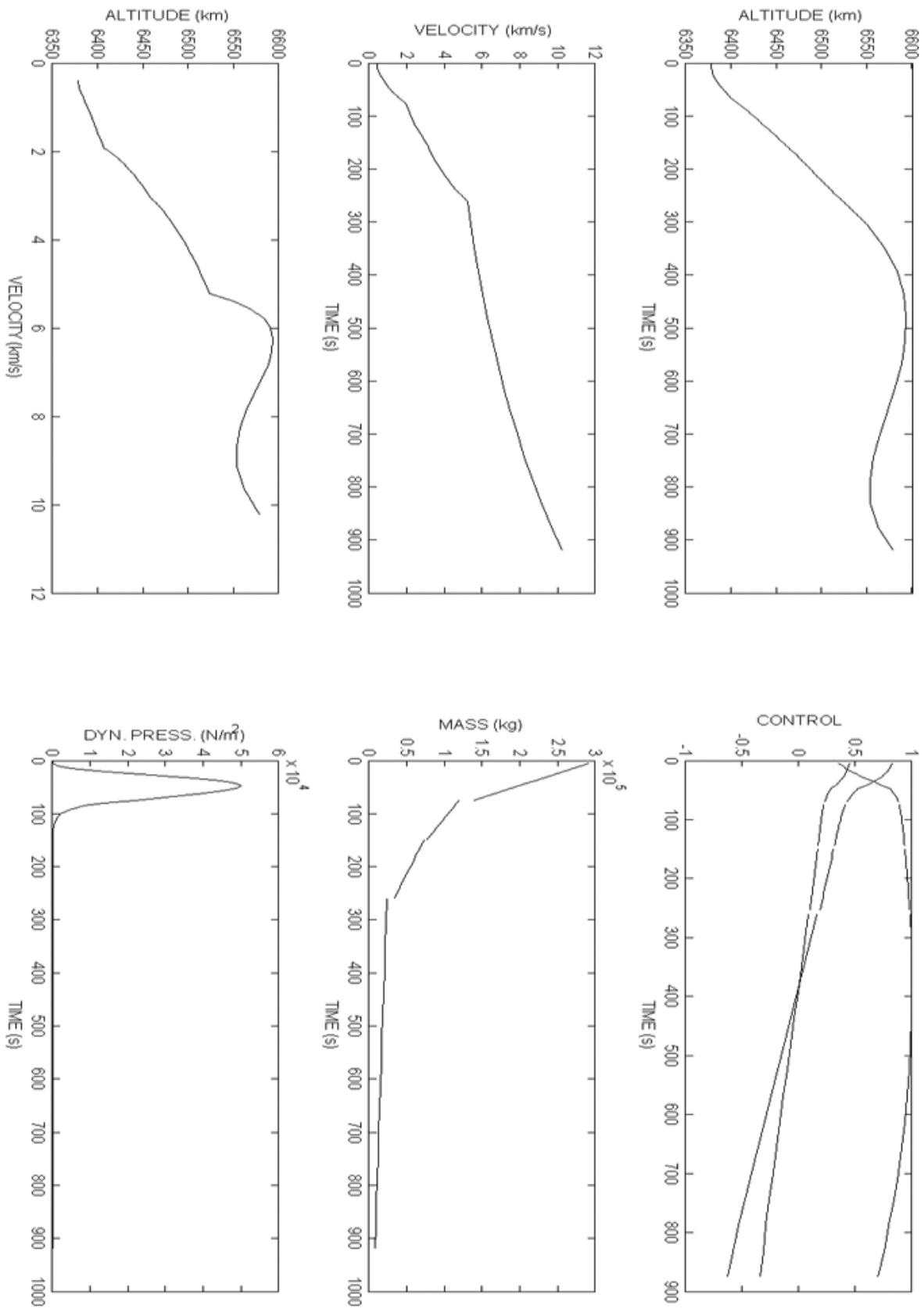

Figure 6.6: Illustration of the control and physical behavior of the launch vehicle

6.6.5 Launch Vehicle: Conclusions and Interpretation

The optimal control of a multiple-stage launch vehicle has been considered. In contrast to the original purely continuous approach in GPOPS [37], here some mixed integer extensions regarding the booster configuration along with further non-linear constraints have been added. Analyzing the impact of those discrete aspects (Table 6.21) revealed a non-intuitive and diverse integer complexity. The resulting MINLP problem can therefore be categorized as very challenging and consists of well over 100 variables and constraints, which is at the limit of current state of the art MINLP solvers, based on evolutionary algorithms.

Two numerical test series of 30 runs each applied a hybrid strategy of the stochastic MIDACO algorithm coupled with a deterministic SQP method to solve the MINLP problem. In both test series MIDACO was able to provide the best known integer combination of the MINLP with high probability, while the SQP method was in most cases able to successfully refine those MIDACO solutions in a reasonable time of about 5 Minutes. Interestingly it could be observed, that a relative small cpu-time budget of 600 (10 Minutes) seconds for MIDACO was sufficient, to obtain premature MINLP solutions of such quality, that the SQP algorithm could always successfully refine those. In contrast to this, the larger budget of 7200 seconds had two undesired effects, which is a less probability in the best known integer combination and some over-fitting that led to premature convergence of the SQP method. The lower probability in finding the best known integer seems to be correlated to the amount of $\{9, 3, -\}$ combinations revealed by MIDACO. Due to the insignificance of the third integer variable T_2 in the scenario of $B_1 = 0$, those solutions have a five times higher probability. As the refinement of the solution to the optimal control problem is highly depending on the precision of the continuous variables, a longer

runtime for MIDACO implies therefore a higher probability to switch to the sub-optimal $\{9, 3, -\}$.

Conclusions

In this thesis a conceptual new algorithm for general MINLP was developed. It is based on a combination of an extension of the ACO metaheuristic (Chapter 2) and the Oracle Penalty Method (Chapter 3) for constraint handling. Both components are novel developments that contribute to the field of heuristic optimization theory and can be found in separate publications in the open literature in [43], [44] and [45]. A sophisticated implementation (MIDACO) of this new MINLP algorithm was engineered (Chapter 4) and has already been adopted by multiple international academic institutions (see the Introduction or [28] and [49], where MIDACO is applied in the open literature), stating the practical relevance of the here developed work.

Due to the heuristic nature of the here proposed algorithm, a comprehensive amount of numerical results has been presented in order to evaluate the usefulness and performance of this novel approach. The numerical results were divided into two major parts: Chapter 5 focused on the performance on comprehensive MINLP benchmark sets with up to 100 test instances and compared the MIDACO performance with those of several established MINLP software codes (in esp. MISQP, BONMIN and COUENNE). It was revealed that the here proposed approach is absolute competitive with the established ones (see Table 5.5 and Table 5.6). In terms of number of global optimal solutions and cpu runtime

performance it was even able to outperform its competitors, based on classical MINLP algorithms as illustrated in Chapter 1. Regarding the number of function evaluation (which is an important criteria for cpu time expensive applications), it could be shown, that the implementation is also competitive, if (massive) parallelization capabilities are available (see Table 5.7). Furthermore, to the best knowledge of the author, this (massive) parallelization capability based on reverse communication is an unique feature of the MIDACO software in the area of MINLP solvers and is a very promising property in respect to the growing trend of parallelization in computer science.

Chapter 6 presented numerical results on real world applications, which were focused on continuous and mixed integer space and aerospace applications. All those applications are known to be difficult or even very difficult. The here developed approach was able to solve those applications to their best known solution and did even reveal several new best known solutions (see Table 6.1). In particular, the concept of mixed integer based optimization was adapted to space applications like the Launch Vehicle (Section 6.6) and Interplanetary Space Mission (Section 6.5) for the very first time here, demonstrating the potential of the introduced approach to MINLP in this context. Furthermore, the hybridization capabilities of the here proposed stochastic algorithm with deterministic algorithms like SQP were demonstrated on the F8-Aircraft (see Table 6.2) and Launch Vehicle (see Table 6.22) application.

In conclusion, this thesis does offer a valuable contribution to both, the theory of heuristic optimization for MINLP in general and the application to real world optimization problems, which were focused here on space applications in particular. With already multiple independent international academic users of the MIDACO software (which expresses the here introduced algorithm) the here developed new approach to MINLP is considered successful.

Appendix A

Evaluation of the Oracle Penalty Method

A set of 60 constrained benchmark problems from the open literature has been considered to evaluate the performance of the penalty functions considered in Chapter 3. Detailed information on the problems is listed in Table 25 while Table 24 contains explanations for the abbreviations used. Further details on those problems can be found in Schittkowski [40].

Table 24: Abbreviations for Table 25

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

Table 25: Information on the benchmark problems

Name	n	n_{int}	m	m_e	$f(x^*, y^*)$
asaadi 1.1	4	3	3	0	-0.409566D+02
asaadi 1.2	4	4	3	0	-0.380000D+02
asaadi 2.1	7	4	4	0	0.694903D+03
asaadi 2.2	7	7	4	0	0.700000D+03
asaadi 3.1	10	6	8	0	0.372195D+02
asaadi 3.2	10	10	8	0	0.430000D+02
van de Braak 1	7	3	2	0	0.100000D+01
van de Braak 2	7	3	4	0	-0.271828D+01
van de Braak 3	7	3	4	0	-0.898000D+02

Table 26: Information on the benchmark problems (continued)

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

Table 29 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 27 explains all abbreviations used in Table 29. All results were calculated on the same personal computer with 2 GHz clock rate and 2 GB RAM working memory.

Table 27: Abbreviations for Table 29

Abbreviation	Explanation
Name	Problem name used in the literature
Penalty	Penalty function used in MIDACO
Opt	Number of global optimal solutions found out of 100 test runs
Feas	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 28: Detailed results for the constrained benchmark problems

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
asaadi 1.1	Oracle_{update}	100	100	-40.95	-40.95	-40.95	1642	0.014
	Oracle_{fix}	100	100	-40.95	-40.95	-40.95	1144	0.015
	Static	100	100	-40.95	-40.95	-40.95	1125	0.013
	Death	100	100	-40.95	-40.95	-40.95	1404	0.013
	Adaptive_1	100	100	-40.95	-40.95	-40.95	1093	0.013
	Adaptive_2	100	100	-40.95	-40.95	-40.95	1051	0.013
	Adaptive_3	100	100	-40.95	-40.95	-40.95	1115	0.012
asaadi 1.2	Oracle_{update}	100	100	-38.00	-38.00	-38.00	78	0.010
	Oracle_{fix}	100	100	-38.00	-38.00	-38.00	72	0.010
	Static	100	100	-38.00	-38.00	-38.00	74	0.009
	Death	100	100	-38.00	-38.00	-38.00	222	0.008
	Adaptive_1	100	100	-38.00	-38.00	-38.00	73	0.009
	Adaptive_2	100	100	-38.00	-38.00	-38.00	72	0.008
	Adaptive_3	100	100	-38.00	-38.00	-38.00	73	0.008
asaadi 2.1	Oracle_{update}	100	100	694.90	694.97	694.94	8040	0.041
	Oracle_{fix}	100	100	694.90	694.97	694.94	1523	0.014
	Static	100	100	694.90	694.97	694.94	1804	0.014
	Death	100	100	694.90	694.97	694.94	1504	0.013
	Adaptive_1	100	100	694.90	694.97	694.94	1563	0.014
	Adaptive_2	100	100	694.90	694.97	694.94	1040	0.012
	Adaptive_3	100	100	694.90	694.97	694.94	1014	0.011

Table 29: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
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	10520	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
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

Table 30: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
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

Table 31: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
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
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

Table 32: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
nvs20	Oracle_{update}	2	100	230.94	307.13	262.93	158944	1.140
	Oracle_{fix}	0	100	231.18	259.88	242.66	160000	1.121
	Static	3	100	230.93	259.24	240.98	157726	1.139
	Death	0	100	242.73	493.01	319.40	160000	0.944
	Adaptive_1	4	100	230.93	259.21	240.24	158479	1.146
	Adaptive_2	14	100	230.94	244.60	239.29	153730	1.123
	Adaptive_3	2	100	230.94	244.50	240.87	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	181.33	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
nvs23	Oracle_{update}	100	100	-1125.2	-1125.2	-1125.2	14239	0.092
	Oracle_{fix}	100	100	-1125.2	-1125.2	-1125.2	2787	0.025
	Static	100	100	-1125.2	-1125.2	-1125.2	1918	0.019
	Death	100	100	-1125.2	-1125.2	-1125.2	4365	0.031
	Adaptive_1	100	100	-1125.2	-1125.2	-1125.2	1673	0.017
	Adaptive_2	100	100	-1125.2	-1125.2	-1125.2	1874	0.019
	Adaptive_3	100	100	-1125.2	-1125.2	-1125.2	1997	0.020
nvs24	Oracle_{update}	48	100	-1033.2	-1030.8	-1032.4	71419	0.489
	Oracle_{fix}	98	100	-1033.2	-1032.0	-1033.1	21248	0.152
	Static	100	100	-1033.2	-1033.2	-1033.2	16061	0.115
	Death	99	100	-1033.2	-1032.0	-1033.1	17360	0.119
	Adaptive_1	99	100	-1033.2	-1032.0	-1033.1	16065	0.115
	Adaptive_2	100	100	-1033.2	-1033.2	-1033.2	13610	0.098
	Adaptive_3	100	100	-1033.2	-1033.2	-1033.2	16820	0.121
duran/	Oracle_{update}	100	100	6.009	6.010	6.010	10001	0.040
grossmann 1	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/	Oracle_{update}	27	97	73.030	145.56	80.393	95846	0.472
grossmann 2	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	112.06	110.90	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

Table 33: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
duran/	Oracle_{update}	16	64	68.006	98.877	70.987	154559	1.103
grossmann 3	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
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	-169983	-169993	916	0.010
	Oracle_{fix}	100	100	-170000	-169983	-169991	382	0.009
	Static	100	100	-170000	-169983	-169989	539	0.009
	Death	100	100	-170000	-169983	-169990	986	0.010
	Adaptive_1	100	100	-170000	-169983	-169990	471	0.009
	Adaptive_2	100	100	-170000	-169983	-169992	476	0.008
	Adaptive_3	100	100	-170000	-169983	-169990	500	0.008

Table 34: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
ST_E36	Oracle_{update}	0	100	-243.85	-147.00	-184.57	20000	0.045
	Oracle_{fix}	0	100	-243.85	-198.79	-224.98	20000	0.043
	Static	0	100	-243.85	-198.79	-225.35	20000	0.042
	Death	0	100	-237.68	-166.50	-177.55	20000	0.040
	Adaptive_1	0	100	-237.68	-166.50	-178.21	20000	0.040
	Adaptive_2	0	99	-237.68	-166.50	-196.98	20000	0.042
	Adaptive_3	0	100	-237.68	-166.50	-180.75	20000	0.040
ST_E38	Oracle_{update}	100	100	7196.87	7198.43	7197.74	4156	0.018
	Oracle_{fix}	100	100	7197.08	7198.44	7198.16	10413	0.029
	Static	100	100	7197.06	7198.44	7198.13	5664	0.020
	Death	0	100	7200.07	7446.94	7347.08	40000	0.087
	Adaptive_1	100	100	7196.99	7198.44	7198.13	4633	0.019
	Adaptive_2	100	100	7197.04	7198.43	7197.94	2185	0.013
	Adaptive_3	100	100	7196.95	7198.44	7197.97	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	281.0	281.0	281.0	25	0.008
	Oracle_{fix}	100	100	281.0	281.0	281.0	23	0.008
	Static	100	100	281.0	281.0	281.0	22	0.008
	Death	100	100	281.0	281.0	281.0	45	0.008
	Adaptive_1	100	100	281.0	281.0	281.0	23	0.009
	Adaptive_2	100	100	281.0	281.0	281.0	20	0.009
	Adaptive_3	100	100	281.0	281.0	281.0	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	-4574.04	-4.000	-4355.22	37472	0.093
	Oracle_{fix}	0	100	-4573.17	-2788.50	-4302.06	60000	0.142
	Static	3	100	-4573.61	-4508.46	-4566.23	58954	0.163
	Death	0	100	0.000	0.000	0.000	60000	0.095
	Adaptive_1	4	100	-4574.01	-4.000	-1310.77	57954	0.133
	Adaptive_2	100	100	-4573.97	-4573.54	-4573.67	9351	0.032
	Adaptive_3	97	100	-4574.03	-4573.38	-4573.69	7507	0.027

Table 35: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
ST_MIQP5	Oracle_{update}	18	100	-333.8	-0.026	-89.37	65417	0.222
	Oracle_{fix}	0	100	-328.1	-159.4	-253.6	70000	0.218
	Static	0	100	-320.4	-132.1	-238.4	70000	0.216
	Death	0	100	-2.947	-0.878	-1.389	70000	0.148
	Adaptive_1	0	100	-303.7	-7.087	-21.99	70000	0.211
	Adaptive_2	0	100	-325.0	-8.102	-185.5	70000	0.221
	Adaptive_3	2	100	-333.8	-161.9	-292.8	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	20019	-28364	216275	2.147
	Oracle_{fix}	0	100	-12747	41715	9559	240000	2.282
	Static	0	100	-16953	28673	5143	240000	2.274
	Death	1	98	-29605	29383	-11421	239486	1.145
	Adaptive_1	3	100	-29605	14247	-22419	237622	1.474
	Adaptive_2	3	100	-29605	-7997	-22628	238601	1.413
	Adaptive_3	4	100	-29605	19728	-24619	235769	1.833

Table 36: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
ST_TEST	Oracle_{update}	96	100	-12.812	-12.771	-12.810	41437	0.171
GR1	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_TEST	Oracle_{update}	5	100	-20.590	-20.220	-20.436	196512	1.567
GR3	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
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

Table 37: Detailed results for the constrained benchmark problems (continued)

Name	Penalty	Opt	Feas	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
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

Appendix B

MIDACO Performance on 100 MINLP Benchmarks

Individual results by MIDACO for a test run on the set of 100 non-convex MINLP benchmarks provided by Schittkowski [40] are reported. Table 38 lists the individual benchmark names as reported in [40] and addresses a library number to them.

Table 38: Benchmark names with corresponding library number

1	MITP1	26	NVS09	51	FLOUDAS5	76	ST TEST2
2	QIP1	27	NVS10	52	FLOUDAS6	77	ST TEST3
3	MITP2	28	NVS11	53	OAER	78	ST TEST4
4	ASAADI11	29	NVS12	54	SPRING	79	ST TEST5
5	ASAADI12	30	NVS13	55	GEAR	80	ST TEST6
6	ASAADI21	31	NVS14	56	DAKOTA	81	ST TEST8
7	ASAADI22	32	NVS15	57	GEAR4	82	ST TESTGR1
8	ASAADI31	33	NVS16	58	GEAR3	83	ST TESTGR3
9	ASAADI32	34	NVS17	59	EX1252A	84	ST TESTPH4
10	DIRTY	35	NVS18	60	EX1263A	85	TLN2
11	BRAAK1	36	NVS19	61	EX1264A	86	TLN4
12	BRAAK2	37	NVS20	62	EX1265A	87	TLN5
13	BRAAK3	38	NVS21	63	EX1266A	88	TLN6
14	DEX2	39	NVS22	64	DU OPT5	89	PROB02
15	CROP5	40	NVS23	65	DU OPT	90	TLOSS
16	TP83	41	NVS24	66	ST E32	91	TLTR
17	WP02	42	GEAR3A	67	ST E36	92	ALAN
18	NVS01	43	WINDFAC	68	ST E38	93	MEANVARX
19	NVS02	44	DG1	69	ST E40	94	HMITTELMANN
20	NVS03	45	DG2	70	ST MIQP1	95	MIP EX
21	NVS04	46	DG3	71	ST MIQP2	96	MGRID CYCLES1
22	NVS05	47	FLOUDAS1	72	ST MIQP3	97	MGRID CYCLES2
23	NVS06	48	FLOUDAS2	73	ST MIQP4	98	CROP20
24	NVS07	49	FLOUDAS3	74	ST MIQP5	99	CROP50
25	NVS08	50	FLOUDAS4	75	ST TEST1	100	CROP100

Based on the results of Table 5.5 (where 10 different runs on the full library where per-

formed) the random *Seed* 4.2 was fixed to 5, which represents a good MIDACO performance. A maximal time budget of 300 seconds (5 minutes) has been assigned (instead of 1000 seconds like in Section 5.2.2). Besides the cpu time budget, the success criteria (5.2) in finding a global optimal solution was applied with $\varepsilon = 0.01$ and $acc = 0.0001$. Table 39 lists the individual results, the abbreviations used are as follows:

$Flag$	-	Symbol indicating a local (-), false (x) or better (o) solution
$Nr.$	-	MINLP problem number from the library
$Eval$	-	Number of function evaluation used by MIDACO
$Time$	-	Cpu time in seconds used by MIDACO
n	-	Number of decision variables in total of the problem
n_{int}	-	Number of integer decision variables of the problem
m	-	Number of constraints in total of the problem
m_{eq}	-	Number of equality constraints of the problem
$f(x^*, y^*)$	-	Global (or best known) optimal objective function value
$f(x, y)$	-	Best (feasible) objective function value obtained by MIDACO
$Violation$	-	Constraint violation measured as L_∞-norm over all violations

Table 39: Individual MIDACO results on 100 MINLP problems

$Flag$	$Nr.$	$Eval$	$Time$	n	n_{int}	m	m_{eq}	$f(x^*, y^*)$	$f(x, y)$	$Violation$
	1	3705	0.0	5	3	1	0	-10009.6900	-9916.3953	0.00000
	2	6	0.0	4	4	4	0	-20.0000	-20.0000	0.00000
	3	20727	0.0	5	3	7	0	3.5000	3.5026	0.00000
	4	13798	0.0	4	3	3	0	-40.9566	-40.6317	0.00000
	5	59	0.0	4	4	3	0	-38.0000	-38.0000	0.00000
	6	4242	0.0	7	4	4	0	694.9027	696.3917	0.00000
	7	89	0.0	7	7	4	0	700.0000	700.0000	0.00000
	8	24443	0.0	10	6	8	0	37.2195	37.5528	0.00000
	9	717	0.0	10	10	8	0	43.0000	43.0000	0.00000
	10	80	0.0	25	13	10	0	-304723942	-301895650	0.00000
	11	31677	0.1	7	3	2	0	1.0000	1.0059	0.00000
	12	27922	0.0	7	3	4	0	-2.7183	-2.6941	0.00000
	13	96735	0.1	7	3	4	0	-8980002	-8948767	0.00000
	14	1	0.0	2	2	2	0	-56.9375	-56.9375	0.00000
	15	228	0.0	5	5	3	0	0.1004	0.1004	0.00000
	16	6247	0.0	5	2	6	0	-30665.5	-30372.7	0.00000
	17	13	0.0	2	1	2	0	-2.4444	-2.4368	0.00000
	18	472043	0.7	3	2	3	1	12.4697	12.4697	0.00007
	19	465482	0.6	8	5	3	3	5.9642	6.0164	0.00006
	20	35	0.0	2	2	2	0	16.0000	16.0000	0.00000
	21	58	0.0	2	2	0	0	0.7200	0.7200	0.00000
-	22	245937264	300.0	8	2	9	4	5.4709	29.4678	0.00008
	23	43	0.0	2	2	0	0	1.7703	1.7703	0.00000
	24	311	0.0	3	3	2	0	4.0000	4.0000	0.00000
	25	8188	0.0	3	2	3	0	23.4497	23.6467	0.00000
	26	148	0.0	10	10	0	0	-43.1343	-43.1343	0.00000
	27	41	0.0	2	2	2	0	-310.8000	-310.8000	0.00000
	28	113	0.0	3	3	3	0	-431.0000	-427.8000	0.00000
	29	119	0.0	4	4	4	0	-481.2000	-481.2000	0.00000
	30	159	0.0	5	5	5	0	-585.2000	-582.8000	0.00000

Table 40: Individual MIDACO results on 100 MINLP problems (continued)

Flag	*Nr.*	*Eval*	*Time*	n	n_{int}	m	m_{eq}	$f(x^*, y^*)$	$f(x, y)$	*Violation*
	31	41158	0.1	8	5	3	3	-40358.1500	-40256.3555	0.00004
	32	157	0.0	3	3	1	0	1.0000	1.0000	0.00000
	33	17	0.0	2	2	0	0	0.7031	0.7031	0.00000
	34	278	0.0	7	7	7	0	-1100.4000	-1093.0000	0.00000
	35	165	0.0	6	6	6	0	-778.4000	-772.4000	0.00000
	36	259	0.0	8	8	8	0	-1098.4000	-1088.6000	0.00000
	37	197257	0.4	16	5	8	0	230.9222	233.1332	0.00000
	38	5690	0.0	3	2	2	0	-5.6848	-5.6746	0.00000
	39	533239	0.7	8	4	9	4	6.0582	6.0582	0.00008
	40	620	0.0	9	9	9	0	-1125.2000	-1114.4000	0.00000
	41	405	0.0	10	10	10	0	-1033.2000	-1023.0000	0.00000
	42	70829	0.1	8	4	4	4	1.0000	1.0024	0.00002
	43	11231785	18.5	14	3	13	13	0.2545	0.2543	0.00008
	44	79347	0.1	6	3	6	0	6.0097	6.0536	0.00000
	45	40632	0.1	11	5	14	1	73.0357	73.7594	0.00008
	46	149590	0.3	17	8	23	2	68.0100	68.6351	0.00008
	47	185537	0.2	5	3	5	2	7.6672	7.6672	0.00008
	48	30249	0.0	3	1	3	0	1.0765	1.0820	0.00009
	49	197477	0.2	7	4	9	0	4.5796	4.6250	0.00000
	50	13470477	18.7	11	8	7	3	-0.9435	-0.9435	0.00008
	51	150	0.0	2	2	4	0	31.0000	31.0000	0.00000
	52	1222	0.0	2	1	3	0	-17.0000	-16.8306	0.00000
	53	126780	0.2	9	3	7	3	-1.9231	-1.9191	0.00004
	54	1204227	2.3	18	12	8	5	0.8462	0.8436	0.00009
	55	3	0.0	4	4	0	0	1.0000	1.0021	0.00000
	56	13930	0.0	4	2	2	0	1.3634	1.3726	0.00000
	57	11501	0.0	6	4	1	1	1.0000	1.0003	0.00007
	58	70829	0.1	8	4	4	4	1.0000	1.0024	0.00002
-	59	120829756	300.0	24	9	34	13	128918.0	134407.2	0.00009
	60	6119043	18.5	24	24	35	0	19.6000	19.6000	0.00000
	61	6431077	19.3	24	24	35	0	8.6000	8.6000	0.00000
	62	40895454	146.9	35	35	44	0	10.3000	10.3000	0.00000
	63	5975068	23.9	48	48	53	0	16.3000	16.3000	0.00000
	64	184918	1.1	20	13	9	0	8.4806	8.5625	0.00000
	65	206575	1.7	20	13	9	0	3.5392	3.5501	0.00000
x	66	87406151	300.0	35	19	18	17	-1.4304	-1.4365	0.02039
	67	69575	0.1	2	1	2	1	-246.0000	-243.8568	0.00003
	68	3718	0.0	4	2	3	0	7197.7271	7203.1783	0.00000
	69	15886	0.0	4	3	5	1	28.2426	28.4142	0.00003
	70	28	0.0	5	5	1	0	281.0000	281.0000	0.00000
	71	4628	0.0	4	4	3	0	2.0000	2.0000	0.00000
	72	21236	0.0	2	2	1	0	-6.0000	-6.0000	0.00000
	73	50668	0.1	6	3	4	0	-4574.0000	-4532.5531	0.00000
	74	205239	0.3	7	2	13	0	-333.8900	-333.7328	0.00009
	75	111	0.0	5	5	1	0	-4500.0000	-4497.5000	0.00000
	76	267	0.0	6	6	2	0	-9.2500	-9.2500	0.00000
	77	116161	0.3	13	13	10	0	-7.0000	-7.0000	0.00000
	78	70826	0.1	6	6	5	0	-7.0000	-7.0000	0.00000
	79	471	0.0	10	10	11	0	-110.0000	-110.0000	0.00000
	80	96	0.0	10	10	5	0	471.0000	471.0000	0.00000

Table 41: Individual MIDACO results on 100 MINLP problems (continued)

Flag	*Nr.*	*Eval*	*Time*	n	n_{int}	m	m_{eq}	$f(x^*, y^*)$	$f(x, y)$	*Violation*
	81	343156	1.0	24	24	20	0	-29605.0	-29311.0	0.00000
	82	3139	0.0	10	10	5	0	-12.8116	-12.6946	0.00000
	83	123064	0.3	20	20	20	0	-20.5900	-20.3932	0.00000
	84	144	0.0	3	3	10	0	-80.5000	-80.5000	0.00000
	85	78	0.0	8	8	12	0	2.3000	2.3000	0.00000
	86	3236690	9.3	24	24	24	0	8.3000	8.3000	0.00000
	87	15715534	53.8	35	35	30	0	10.3000	10.4000	0.00000
	88	26157808	97.3	48	48	36	0	14.6000	14.7000	0.00000
	89	7268	0.0	6	6	8	0	112235.0	112235.0	0.00000
	90	4585740	18.4	48	48	53	0	16.3000	16.3000	0.00000
	91	398513	1.6	48	48	54	0	48.0667	48.0667	0.00000
	92	306034	0.4	8	4	7	2	2.9250	2.9262	0.00003
	93	3132273	10.9	35	14	44	8	14.1897	14.3309	0.00009
	94	232	0.0	16	16	7	0	13.0000	13.0000	0.00000
	95	24412	0.0	5	3	7	0	3.5000	3.5177	0.00000
	96	94	0.0	5	5	1	0	8.0000	8.0000	0.00000
	97	38993	0.1	10	10	1	0	300.0000	302.0000	0.00000
	98	81355	0.4	20	20	3	0	0.1318	0.1329	0.00000
	99	129314	1.4	50	50	3	0	0.4052	0.4090	0.00000
	100	599873	12.1	100	100	3	0	1.0973	1.0975	0.00000

Table 42: Summary of results presented in Table 39

Global optimal Solutions	:	97
Feasible Solutions	:	99
Local Solution (-)	:	2
False Solution (x)	:	1
Average Evaluation	:	1485525
Average Time	:	4.82
Total Time	:	1363.2 [Hour: 0, Min:22, Sec:43]

Appendix C

BONMIN, COUENNE and MIDACO on GAMS Benchmarks

Individual results by the MINLP solvers BONMIN [6], COUENNE [4] and MIDACO (version 2.0) for a set of 66 MINLP problems form the GAMS MINLPlib [22] are presented. A maximal cpu time budget of 300 seconds has been applied to all solvers for each instance. In case of MIDACO the automatic stopping criteria (see Section 4.2) was activated (using $Autostop = 50$). For BONMIN and COUENNE only one test run for every problem was performed, using the pre-defined starting point from the GAMS MINLPlib. MIDACO was tested 10 times with a different random seed on every instance. Therefore in case of MIDACO the number of global optimal (*Optimal*) and feasible (*Feasible*) solutions is reported as a fraction, where the numerator denotes the number of successful runs out of all 10 runs, for every problem. In case BONMIN and COUENNE did not reach the global optimal solution on an instance, the sub-optimal objective function value is reported in brackets. More details on the individual problems (in esp. the global optimal objective function value) can be found in Table 39. A summary of the individual results presented in Table 43 can be found in Section 5.3 Table 5.6.

Table 43: Individual results by BONMIN, COUENNE and MIDACO

Problem	*Solver*	*Optimal*	*Feasible*	*Time*
NVS01	BONMIN	Yes	Yes	0.79
	COUENNE	No (12.8817)	Yes	0.45
	MIDACO	9/10	10/10	0.90
NVS02	BONMIN	Yes	Yes	0.81
	COUENNE	Yes	Yes	0.05
	MIDACO	10/10	10/10	4.83
NVS03	BONMIN	Yes	Yes	0.47
	COUENNE	Yes	Yes	0.03
	MIDACO	10/10	10/10	0.09
NVS04	BONMIN	Yes	Yes	0.55
	COUENNE	Yes	Yes	0.14
	MIDACO	10/10	10/10	0.08
NVS05	BONMIN	Yes	Yes	3.25
	COUENNE	Yes	Yes	39.11
	MIDACO	0/10	10/10	290.37
NVS06	BONMIN	Yes	Yes	0.20
	COUENNE	Yes	Yes	0.03
	MIDACO	10/10	10/10	0.11
NVS07	BONMIN	Yes	Yes	0.19
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.17
NVS08	BONMIN	Yes	Yes	0.29
	COUENNE	No (29.0)	Yes	0.08
	MIDACO	10/10	10/10	0.60
NVS09	BONMIN	Yes	Yes	0.11
	COUENNE	Yes	Yes	0.12
	MIDACO	10/10	10/10	0.52
NVS10	BONMIN	Yes	Yes	0.17
	COUENNE	Yes	Yes	0.08
	MIDACO	10/10	10/10	0.09
NVS11	BONMIN	Yes	Yes	0.17
	COUENNE	No (-426.6)	Yes	0.08
	MIDACO	10/10	10/10	0.15
NVS12	BONMIN	Yes	Yes	0.19
	COUENNE	No (-473.2)	Yes	0.42
	MIDACO	10/10	10/10	0.21
NVS13	BONMIN	Yes	Yes	0.27
	COUENNE	Yes	Yes	1.32
	MIDACO	10/10	10/10	0.27
NVS14	BONMIN	Yes	Yes	0.89
	COUENNE	Yes	Yes	0.08
	MIDACO	10/10	10/10	2.54
NVS15	BONMIN	Yes	Yes	0.22
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.13
NVS16	BONMIN	Yes	Yes	0.25
	COUENNE	Yes	Yes	0.08
	MIDACO	10/10	10/10	0.08

Table 44: Individual results by BONMIN, COUENNE and MIDACO (continued)

Problem	*Solver*	*Optimal*	*Feasible*	*Time*
NVS17	BONMIN	Yes	Yes	0.45
	COUENNE	No (-1096.8)	Yes	124.81
	MIDACO	10/10	10/10	0.58
NVS18	BONMIN	Yes	Yes	0.44
	COUENNE	No (-745.4)	Yes	300.00
	MIDACO	10/10	10/10	0.44
NVS19	BONMIN	Yes	Yes	1.38
	COUENNE	-	No	300.00
	MIDACO	10/10	10/10	0.74
NVS20	BONMIN	No (241.4073)	Yes	0.67
	COUENNE	Yes	Yes	5.79
	MIDACO	10/10	10/10	86.36
NVS21	BONMIN	Yes	Yes	0.79
	COUENNE	Yes	Yes	0.18
	MIDACO	7/10	10/10	0.61
NVS22	BONMIN	Yes	Yes	1.03
	COUENNE	Yes	Yes	0.19
	MIDACO	10/10	10/10	4.78
NVS23	BONMIN	No (-1113.8)	Yes	0.58
	COUENNE	No (-1104.4)	Yes	300.00
	MIDACO	10/10	10/10	0.95
NVS24	BONMIN	No (-1031.8)	Yes	0.52
	COUENNE	-	No	300.00
	MIDACO	10/10	10/10	1.29
WINDFAC	BONMIN	Yes	Yes	0.43
	COUENNE	Yes	Yes	0.61
	MIDACO	1/10	10/10	43.87
OAER	BONMIN	Yes	Yes	0.21
	COUENNE	Yes	Yes	0.12
	MIDACO	10/10	10/10	9.92
SPRING	BONMIN	Yes	Yes	1.25
	COUENNE	Yes	Yes	0.89
	MIDACO	10/10	10/10	137.90
GEAR	BONMIN	Yes	Yes	0.56
	COUENNE	Yes	Yes	0.09
	MIDACO	10/10	10/10	0.13
GEAR3	BONMIN	Yes	Yes	0.85
	COUENNE	Yes	Yes	0.13
	MIDACO	10/10	10/10	4.51
GEAR4	BONMIN	Yes	Yes	300.00
	COUENNE	Yes	Yes	1.79
	MIDACO	10/10	10/10	1.71
EX1252A	BONMIN	No (134471.5605)	Yes	9.13
	COUENNE	Yes	Yes	8.67
	MIDACO	0/10	2/10	300.00
EX1263A	BONMIN	No (21.3)	Yes	6.69
	COUENNE	No (21.3)	Yes	1.09
	MIDACO	3/10	10/10	11.12

Table 45: Individual results by BONMIN, COUENNE and MIDACO (continued)

Problem	*Solver*	*Optimal*	*Feasible*	*Time*
EX1264A	BONMIN	No (9.3)	Yes	10.86
	COUENNE	No (9.0)	Yes	1.53
	MIDACO	2/10	10/10	5.94
EX1265A	BONMIN	No (11.3)	Yes	17.53
	COUENNE	No (10.6)	Yes	4.24
	MIDACO	1/10	10/10	11.09
EX1266A	BONMIN	Yes	Yes	15.67
	COUENNE	Yes	Yes	1.67
	MIDACO	5/10	10/10	38.39
DU OPT5	BONMIN	No (8.3266)	Yes	2.58
	COUENNE	No (10.8967)	Yes	300.00
	MIDACO	10/10	10/10	106.31
DU OPT	BONMIN	Yes	Yes	2.07
	COUENNE	No (9.3672)	Yes	300.00
	MIDACO	4/10	10/10	206.63
ST E32	BONMIN	Yes	Yes	1.45
	COUENNE	Yes	Yes	21.06
	MIDACO	0/10	0/10	300.00
ST E36	BONMIN	Yes	Yes	0.20
	COUENNE	Yes	Yes	0.18
	MIDACO	4/10	10/10	0.54
ST E38	BONMIN	Yes	Yes	0.13
	COUENNE	Yes	Yes	0.11
	MIDACO	10/10	10/10	1.07
ST E40	BONMIN	-	No	0.04
	COUENNE	Yes	Yes	0.32
	MIDACO	10/10	10/10	0.46
ST MIQP1	BONMIN	Yes	Yes	0.37
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.15
ST MIQP2	BONMIN	Yes	Yes	0.53
	COUENNE	Yes	Yes	0.08
	MIDACO	10/10	10/10	0.50
ST MIQP3	BONMIN	Yes	Yes	0.17
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.41
ST MIQP4	BONMIN	Yes	Yes	0.25
	COUENNE	Yes	Yes	0.06
	MIDACO	5/10	10/10	2.62
ST MIQP5	BONMIN	Yes	Yes	0.17
	COUENNE	Yes	Yes	0.09
	MIDACO	10/10	10/10	6.51
ST TEST1	BONMIN	Yes	Yes	0.33
	COUENNE	Yes	Yes	0.09
	MIDACO	10/10	10/10	0.51
ST TEST2	BONMIN	Yes	Yes	0.12
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.61

Table 46: Individual results by BONMIN, COUENNE and MIDACO (continued)

Problem	*Solver*	*Optimal*	*Feasible*	*Time*
ST TEST3	BONMIN	Yes	Yes	0.76
	COUENNE	Yes	Yes	0.08
	MIDACO	10/10	10/10	3.13
ST TEST4	BONMIN	-	No	0.03
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	1.04
ST TEST5	BONMIN	Yes	Yes	1.29
	COUENNE	Yes	Yes	0.23
	MIDACO	10/10	10/10	0.41
ST TEST6	BONMIN	Yes	Yes	0.83
	COUENNE	Yes	Yes	0.11
	MIDACO	10/10	10/10	0.41
ST TEST8	BONMIN	No (-29575.0)	Yes	0.11
	COUENNE	Yes	Yes	0.14
	MIDACO	10/10	10/10	11.35
ST TESTGR1	BONMIN	No (-12.7758)	Yes	0.78
	COUENNE	No (-12.7842)	Yes	0.08
	MIDACO	10/10	10/10	1.07
ST TESTGR3	BONMIN	No (-19.9724)	Yes	0.14
	COUENNE	No (-20.4910)	Yes	0.07
	MIDACO	8/10	10/10	6.15
ST TESTPH4	BONMIN	Yes	Yes	0.16
	COUENNE	Yes	Yes	0.06
	MIDACO	10/10	10/10	0.16
TLN2	BONMIN	Yes	Yes	5.44
	COUENNE	Yes	Yes	0.11
	MIDACO	10/10	10/10	0.66
TLN4	BONMIN	No (8.5)	Yes	140.56
	COUENNE	Yes	Yes	37.02
	MIDACO	2/10	10/10	5.20
TLN5	BONMIN	No (10.9)	Yes	300.00
	COUENNE	No (10.6)	Yes	300.00
	MIDACO	1/10	10/10	13.27
TLN6	BONMIN	No (19.0)	Yes	300.00
	COUENNE	No (15.8)	Yes	300.00
	MIDACO	0/10	10/10	54.03
PROB02	BONMIN	Yes	Yes	0.14
	COUENNE	Yes	Yes	0.09
	MIDACO	10/10	10/10	0.68
TLOSS	BONMIN	Yes	Yes	20.98
	COUENNE	Yes	Yes	4.74
	MIDACO	3/10	10/10	33.29
TLTR	BONMIN	Yes	Yes	20.25
	COUENNE	Yes	Yes	2.84
	MIDACO	10/10	10/10	45.40
ALAN	BONMIN	Yes	Yes	0.29
	COUENNE	Yes	Yes	0.23
	MIDACO	10/10	10/10	7.93

Table 47: Individual results by BONMIN, COUENNE and MIDACO (continued)

Problem	*Solver*	*Optimal*	*Feasible*	*Time*
MEANVARX	BONMIN	No (14.5147)	Yes	0.22
	COUENNE	Yes	Yes	1.91
	MIDACO	9/10	10/10	300.00
HMITTELMANN	BONMIN	Yes	Yes	9.38
	COUENNE	Yes	Yes	0.36
	MIDACO	10/10	10/10	0.74

Bibliography

[1] M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optim. Eng.*, 5(2):157–177, 2004.

[2] M. A. Abramson. Nomadm optimization software. Software available at http://www.gerad.ca/NOMAD/Abramson/nomadm.html, 2011.

[3] M. A. Abramson, C. Audet, J. Chrissis, and J. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Opt. Lett.*, 3(1):35–47, 2009.

[4] COIN-OR (Project Manager P. Belotti). Convex over and under envelopes for nonlinear estimation. Software available at http://www.coin-or.org/Couenne/, 2011.

[5] D. A. Benson. *A Gauss Pseudospectral Transcription for Optimal Control*. PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 2004.

[6] COIN-OR (Project Manager P. Bonami). Basic open-source nonlinear mixed integer programming. Software available at http://www.coin-or.org/Bonmin/, 2011.

[7] G. E. P. Box and M. E. Mueller. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 29(2):610–611, 1958.

[8] M. R. Bussieck and S. Vigerske. *MINLP Solver Software*. John Wiley and Sons, Inc., 2010.

[9] B.K.S. Cheung, A. Langevin, and H. Delmaire. Coupling genetic algorithm with a grid search method to solve mixed integer nonlinear programming problems. *Comput. Math. Appl.*, 34(12):13 – 23, 1997.

[10] C.A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Comput. Method. Appl. M.*, 191(11):1245–1287, 2002.

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

[12] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano (Italy), 1992.

[13] M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[14] M. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.*, 36(3):307–339, 1986.

[15] European Space Agency (ESA) and Advanced Concepts Team (ACT). Gtop database - global optimisation trajectory problems and solutions. Software available at http://www.esa.int/gsp/ACT/inf/op/globopt.htm, 2011.

[16] O. Exler, L. T. Antelo, J. A. Egea, A. A. Alonso, and J. R. Banga. A tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. *Comput. Chem. Eng.*, 32(8):1877–1891, 2008.

[17] O. Exler, T. Lehmann, and K. Schittkowski. A comparative study of SQP-type algorithms for nonlinear and nonconvex mixed-integer optimization. *submitted*, 2011.

[18] O. Exler and K. Schittkowski. A trust region SQP algorithm for mixed-integer nonlinear programming. *Opt. Lett.*, 1(3):269–280, 2007.

[19] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Math. Program.*, 66(1):327–349, 1994.

[20] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Math. Program.*, 91:239–270, 2002.

[21] GAMS. General algebraic modeling system. Software available at http://www.gams.com/, 2011.

[22] GAMS. MINLPlib - a collection of mixed integer nonlinear programming models. Software available at http://www.gamsworld.org/minlp/minlplib.htm, 2011.

[23] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1978.

[24] A. M. Geoffrion. Generalized benders decomposition. *J. Optimiz. Theory App.*, 10(4):237–260, 1972.

[25] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[26] I. E. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optim. Eng.*, 3(3):227–252, 2002.

[27] A.B. Hadj-Alouane and J.C. Bean. A genetic algorithm for the multiple choice integer program. *Oper. Res.*, 45(3):92–101, 1997.

[28] M. Haenel, S. Kuhn, D. Henrich, L. Gruene, and J. Pannek. Optimal camera placement to measure distances conservativly regarding static and dynamic obstacles. Article available at http://arxiv.org/abs/1105.3270, 2011.

[29] M.R. Hestenes. Multiplier and gradient methods. *J. Optimiz. Theory App.*, 4:303–320, 1969.

[30] A. Homaifar, C.X. Qi, and S.H. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.

[31] G. T. Huntington. *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control Problems.* PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 2007.

[32] Y. Kaya, C. and L. Noakes, J. Computational algorithm for time-optimal switching control. *J. Optimiz. Theory App.*, 117(1):69–92, 2003.

[33] Dennis Jr. J. E. Kokkolaras M., Audet C. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optim. Eng.*, 2(1):5–29, 2000.

[34] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[35] G. Marsaglia. Xorshift RNGs. *J. Stat. Softw.*, 8(14):1–6, 2003.

[36] A. Munawar, M. Wahib, M. Munetomo, and K. Akama. Advanced genetic algorithm to solve minlp problems over gpu. In *Evolutionary Computation (CEC), 2011 IEEE Congress*, pages 318–325, 2011.

[37] A. V. Rao, D. A. Benson, C. L. Darby, M. A. Patterson, C. Francolin, and G. T. Huntington. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM T. Math. Software*, 37(2):1–39, 2010.

[38] G. Rudolph. *Convergence Properties of Evolutionary Algorithms.* Kovac, 1997.

[39] S. Sager. mintOC: Benchmark library of mixed-integer optimal control problems. Software available at http://mintoc.de, 2011.

[40] K. Schittkowski. A collection of 100 test problems for nonlinear mixed-integer programming in fortran (user guide). Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2009.

[41] K. Schittkowski. NLPQLP - a fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search (user guide). Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2009.

[42] M. Schlueter. MIDACO - Global Optimization Software for Mixed Integer Nonlinear Programming. Software available at http://www.midaco-solver.com, 2012.

[43] M. Schlueter, J. A. Egea, and J. R. Banga. Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Comput. Oper. Res.*, 36(7):2217–2229, 2009.

[44] M. Schlueter, J. A. Egea, Antelo L.T., Alonso A.A., and J. R. Banga. An extended ant colony optimization algorithm for integrated process and control system design. *Ind. Eng. Chem.*, 48(14):6723–6738, 2009.

[45] M. Schlueter and M Gerdts. The oracle penalty method. *J. Global Optim.*, 47(2):293–325, 2010.

[46] M. Schlueter, M. Gerdts, and Rueckmann J.J. Non-linear mixed-integer-based optimisation technique for space applications, ESA NPI Day (poster). Poster available at http://www.midaco-solver.com/download_files/ESA_NPI_Poster.jpg, 2010.

[47] M. Schlueter, M. Gerdts, and Rueckmann J.J. A numerical study of MIDACO on 100 MINLP benchmarks. *Optimization (DOI:10.1080/02331934.2012.668545)*, 2012.

[48] M. Socha, K. Dorigo. Ant colony optimization for continuous domains. *Eur. J. Oper.*, 185(3):1155–1173, 2008.

[49] A.T. Takano and B.G. Marchand. Optimal constellation design for space based situational awareness applications (AAS 11-543). Article available at http://www.ae.utexas.edu/~marchand/AAS11-543.pdf, 2011.

[50] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, 2002.

[51] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Marti. Scatter search and local nlp solvers: A multistart framework for global optimization. *Informs J. on Computing*, 19(3):328–340, July 2007.

[52] T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex minlp problems. *Comput. Chem. Eng.*, 19(1):131 – 136, 1995.

[53] O. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Math. Comput. Appl.*, 10(1):45–56, 2005.

[54] L. Yiqing, Y. Xigang, and L. Yongjian. An improved pso algorithm for solving non-convex nlp/minlp problems with equality constraints. *Comp. Chem. Eng.*, 31(3):153 – 162, 2007.