Extended Ant Colony Optimization for non-convex Mixed Integer Nonlinear Programming

Martin Schlüter$^+$, Jose A. Egea*, Julio R. Banga*
martin.schlueter@uni-bayreuth.de, jegea@iim.csic.es, julio@iim.csic.es
$^+$ *Department of Computer Science, University of Bayreuth*
95440 Bayreuth, Germany
* *Process Engineering Group, Instituto de Investigaciones Marinas (IIM-CSIC)*
36208 Vigo, Spain

19th August 2008

Abstract

Two novel extensions for the well known Ant Colony Optimization (ACO) framework are introduced here, which allow the solution of Mixed Integer Nonlinear Programs (MINLP). Furthermore, a hybrid implementation (`ACOmi`) based on this extended ACO framework, specially developed for complex non-convex MINLPs, is presented together with numerical results.

These extensions on the ACO framework have been developed to serve the needs of practitioners who face highly non-convex and computationally costly MINLPs. The performance of this new method is evaluated considering several non-convex MINLP benchmark problems and one real-world application. The results obtained by our implementation substantiate the success of this new approach.

Keywords: Ant Colony Optimization, MINLP, Global Optimization, Hybrid Metaheuristics, Constrained Optimization, Oracle Penalty Method.

1 Introduction

The first optimization algorithms inspired by ants foraging behavior were introduced by Marco Dorigo in his PhD thesis (Dorigo [12]). Later, these algorithms were formalized as the Ant Colony Optimization (ACO) metaheuristic (Dorigo and Di Caro [13]). Originally the ACO metaheuristic was considered only for combinatorial optimization problems (e.g. Travelling Salesman Problem, Stützle and Dorigo [33]). In Bonabeau et al. [6] a general overview on ACO and its applications on some scientific and engineering problems is given. An introduction to ACO together with recent trends is given in Blum [4]. Comprehensive information on ACO can be found in Dorigo and Stuetzle [14]

Several extensions of the ACO metaheuristic for continuous search domains can be found in the literature, among them Socha and Dorigo [32], Yu et al. [36], Dreo and Siarry [15] or Kong and Tian [27]. Other applications of ACO frameworks for real-world problems, arising from engineering design applications, can be found in Jayaraman et al. [25], Rajesh et al. [29], Chunfeng and Xin [10] or Zhang et al. [37]. In contrast extensions for mixed integer search domains are very rare in the literature. In Socha [31] a general extension on continuous and mixed integer domains is discussed.

Although a detailed explanation of an ACO algorithm design for continuous problems together with numerical results is given in this reference, the application on mixed integer domains is only mentioned theoretically. The proposed approach is a combination of a conventional ACO algorithm, which is based on the concept of a pheromone table for discrete domains, with an extension for continuous domains based on the use of pheromone controlled probability density functions. In Serban and Sandou [30] a mixed integer programming method based on the ACO metaheuristic is introduced especially for the unit commitment problem. Again this reference couples a conventional pheromone table based ACO algorithm for discrete variables with an extension for continuous variables.

This paper introduces a conceptual new extension of the ACO metaheuristic for mixed integer search domains. In contrast to a pheromone table, a pheromone guided discretised probability density function will be used for the discrete search domain. This approach allows an intuitive handling of integer variables besides continuous ones within the same algorithm framework. Whilst the above mentioned extensions of ACO on mixed integer domains can be seen as based on a discrete ACO algorithm with an extension for continuous domains, our approach works the other way round. Based on a continuous ACO methodology we extend the algorithm on discrete variables. This is done by a heuristic, defining a lower bound for the standard deviation of the discretized Gaussian probability function, which is assumed here as probability density function.

To apply the ACO metaheuristic on general MINLPs, not only we had to consider mixed integer search domains, but also a good handling of arbitrary constraints. Here we propose a new penalty strategy that reinforces this approach and which fits very well in the extended ACO metaheuristic. Our method is based on a user-given oracle information, an estimated preferred objective function value, which is used as the crucial parameter for the penalty function. A detailed investigation of this approach is in preparation and the results seem to be promising. In particular the method seems to be quite robust against 'bad' selected oracles. Furthermore it can be shown analytically, that a sufficiently large or sufficiently low oracle can be used as default parameters.

Our implementation, named `ACOmi`, is based on the ACO metaheuristic for continuous domains presented by Socha and Dorigo [32] and enlarged by the above mentioned two novel heuristics. As this implementation is a sophisticated one, aiming on the practical use of real-world applications, several other heuristics (which will be briefly described in Section 5) are included and a mixed integer sequential quadratic programming algorithm is embedded as a local solver in the metaheuristic framework. In addition a set of academic benchmark test problems, a complex MINLP application, a thermal insulation system described in Abramson [1], is solved with this implementation to fortify its practical relevance for real-world applications.

This paper is structured as follows: first, we give a brief overview on the state-of-the-art for solving non-convex MINLPs. Next, we present our mixed integer extension to the ACO framework and introduce a new penalty method in order to handle this class of problems. A detailed description of our implementation `ACOmi`, incorporating this extended ACO framework and the penalty method, is given. The performance of our approach is then evaluated considering (i) a set of MINLP benchmark problems, and (ii) a complex engineering design problem formulated as a MINLP. Finally, we present a set of conclusions.

2 Approaches for non-convex Mixed Integer Nonlinear Programs

MINLPs are the most general type of single-objective optimization problems. Containing both continuous and integer decision variables, and without any limitation to the complexity of either the objective function or the constraints, these problems can be a real challenge. The presence of nonlinearities in the objective and constraint functions might imply non-convexity in MINLP problems, i.e. the potential existence of multiple local solutions.

Before giving an overview of the existing methodologies for such problems, the mathematical formulation of a MINLP is given in (1).

$$\begin{array}{llll}
\text{Minimize} & f(x,y) & & (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{N}^{n_{int}},\ n_{con},\ n_{int} \in \mathbb{N}) \\
\\
\text{subject to:} & g_i(x,y) & = 0, & i = 1,...,m_{eq} \in \mathbb{N} \\
 & g_i(x,y) & \geq 0, & i = m_{eq}+1,...,m \in \mathbb{N} \\
 & x_l \leq x & \leq x_u & (x_l,\ x_u \in \mathbb{R}^{n_{con}}) \\
 & y_l \leq y & \leq y_u & (y_l,\ y_u \in \mathbb{N}^{n_{int}})
\end{array} \tag{1}$$

In this formulation $f(x,y)$ is the objective function, which has to be minimized, depending on x, the vector of n_{con} continuous decision variables, and y, the vector of n_{int} integer decision variables. The functions $g_1,...,g_{m_{eq}}$ represent the equality constraints and the functions $g_{m_{eq}+1},..,g_m$ the inequality constraints. The vectors x_l, x_u and y_l, y_u are the lower and upper bounds for the decision variables x and y, those are also called box-constraints.

In principle two types of approaches are possible to solve this kind of problem: deterministic and stochastic methods. So-called metaheuristics (Glover and Kochenberger [21]) often belong to the latter. ACO can be classified as a stochastic metaheuristic. Modern algorithms, like the one discussed in this paper, often combine both methodologies in a hybrid-manner, consisting of a stochastic framework with deterministic strategies embedded. For example Egea et al. [17], Chelouah and Siarry [9] or Chelouah and Siarry [8] follow also this hybrid framework approach.

Among the deterministic approaches for MINLPs, Branch and Bound techniques, Outer Approximation, General Benders Decomposition or extended Cutting Plane methods are the most common ones. A comprehensive review on these can be found in Grossman [22]. The big advantage of deterministic approaches is that a lot of these can guarantee global optimality. On the other hand this guarantee comes with a disadvantage, the possibility of a tremendous computation time depending on the problem structure. In addition, most of the implementations of these methods require a user given formulation of the mathematical MINLP in an explicit way. In this case the implementation is called a white box solver. In principle any implementation can gain the required information via approximation by function evaluations from a black box formulation alternatively, but this does highly increase the computational time effort.

In contrast to the white box approach, black box solvers do not require any knowledge of the mathematical formulation of the optimization problem. Of course a mathematical formulation is always essential to implement and tackle an optimization problem, but black box solvers do not assimilate this formulation. This property makes them very flexible according to programming languages and problem types, which is very much appreciated by practitioners. All metaheuristics can be seen as black box solvers regarding their fundamental concept. In this paper an extension of the ACO metaheuristic will be introduced to apply this method on general MINLPs. Besides academic benchmark MINLP test problems, one complex engineering application will be considered and optimized with our ACO implementation.

3 The Ant Colony Optimization framework

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 trail will attract other ants to follow it in the hope of finding food again. Over time the pheromones will start to evaporate and therefore reduce the attraction of the path, so only paths that are updated frequently with new pheromones remain attractive. Short paths from the nest to a food source imply short marching times for the ants, so those paths are updated with pheromones more often than long ones. Consequently more and more ants will be attracted by the shorter paths with ongoing time. As a final result, a very short path will be discovered by the ant colony.

This basic idea of ACO algorithms is to mimic this biological behavior with artificial ants 'walking' on a graph, which represents a mathematical problem (e.g. Traveling Salesman Problem). An optimal path in terms of length or some other cost-resource is requested in those problems, which belong to the field of combinatorial optimization. By using a parametrized probabilistic model, called pheromone table, the artificial ants choose a path through a completely connected Graph $G(C, L)$, where C is the set of vertices and L is the set of connections. The set C of vertices represent the solution components, which every ant chooses incrementally to create a path. The pheromone values within the pheromone table are used by the ants, to make these probabilistic decisions. By updating the pheromone values according to information gained on the search domain, this algorithmic procedure leads to very good and hopefully global optimal solutions, like the biological counterpart.

Algorithm 1 *ACO metaheuristic*

```
while stopping criteria not met do
  pheromone based solution construction
  pheromone update
  daemon actions
end while
```

The pseudo-code in Algorithm 1 illustrates this fundamental working procedure of the ACO metaheuristic. The stopping criteria and daemon actions are a choice of the algorithm designer. Commonly used stopping criteria are for example a maximal limit of constructed solutions or a maximal time budget. Daemon actions might be any activity, that cannot be performed by single ants. Local search activities and additional pheromone manipulations are examples for such daemon actions, see Blum [5].

In contrast to the original ACO metaheuristic developed for combinatorial optimization problems, the ACO framework considered in this paper is mainly based on the extended ACO for continuous domains proposed by Socha and Dorigo [32]. The biological visualization of ants choosing their way through a graph-like search domain does not hold any longer for these problems, as these belong to a completely different class. However, the extension of the original ACO metaheuristic to continuous domains is possible without any major conceptual change, see Socha [31]. In this methodology, ACO works by the incremental construction of solutions regarding a probabilistic choice according to a probability density function (PDF), instead of a pheromone table like in the original ACO. In principle any function $P(x) \geq 0$ for all x with the property:

$$\int_{-\infty}^{\infty} P(x)\, dx \;=\; 1 \tag{2}$$

can act as a PDF. Among the most popular functions to be used as a PDF is the Gaussian function. This function has some clear advantages like an easy implementation (e.g. Box and Müller [7]) and a corresponding fast sampling time of random numbers. On the other hand, a single Gaussian function is only able to focus on one mean and therefore not able to describe situations where two or more disjoint areas of the search domain are promising. To overcome this disadvantage by still keeping track of the benefits of a Gaussian function, a PDF $G^i(x)$ consisting of a weighted sum of several one-dimensional Gaussian functions $g^i_l(x)$ is considered for every dimension i of the original search domain:

$$G^i(x) \;=\; \sum_{l=1}^{k} w^i_l \cdot g^i_l(x) \;=\; \sum_{l=1}^{k} w^i_l \frac{1}{\sigma^i_l \sqrt{2\pi}}\, e^{-\frac{(x-\mu^i_l)^2}{2\,\sigma^{i\;2}_l}} \tag{3}$$

This function is characterized by the triplets $(w^i_l, \sigma^i_l, \mu^i_l)$ that are given for every dimension i of the search domain and the number of kernels k of Gauss functions used within $G^i(x)$. Within this triplet, w represents the weights for the individual Gaussian functions for the PDF, σ represents the standard deviations, and μ represents the means for the corresponding Gaussian functions. The indices i and l refer, respectively, to the i-th dimension of the decision vector of the MINLP problem and the l-th kernel number of the individual Gaussian function within the PDF.

As that the above triplets fully characterize the PDF and therefore guide the sampled solution candidates throughout the search domain, they are called pheromones in the ACO sense and constitute the biological background of the ACO metaheuristic presented here. Besides the incremental construction of the solution candidates according to the PDF, the update of the pheromones plays a major role in the ACO metaheuristic.

An obviously good choice to update the pheromones is the use of information, which has been gained throughout the search process so far. This can be done by using a solution archive SA in which the so far most promising solutions are saved. In case of k kernels this can be done choosing an archive size of k. Thus the SA contains k n-dimensional solution vectors s_l and the corresponding k objective function values (see Socha [31]).

As the focus is here on constrained MINLPs, the solution archive SA also contains the corresponding violation of the constraints and the penalty function value for every solution s_l. In particular, the attraction of a solution s_l saved in the archive is measured regarding the penalty function value instead of the objective function value. Details on the measurement of the violation and the penalty function will be described in Section 4.

We now explain the update process for the pheromones which is directly connected to the update process of the solution archive SA. The weights w (which indicate the importance of an ant and therefore rank them) are calculated with a linear proportion according to the parameter k:

$$w^i_l \;=\; \frac{(k-l+1)}{\sum_{j=1}^{k}\, j} \tag{4}$$

With this fixed distribution of the weights, a linear order of priority within the solution archive SA is established. Solutions s_l with a low index l are preferred. Hence, s_1 is the current best solution found and therefore most important, while s_k is the solution of the lowest interest, saved in SA. Updating the solution archive will then directly imply a pheromone update based on best solutions found so far. Every time a new solution (ant) is created and evaluated within a generation its attraction (penalty function value) is compared to the attraction of the so far best solutions saved in SA, starting with the very best solution s_1 and ending up with the last one s_k in the archive. In case the new solution has a better attraction than the j-th one saved in the archive, the new solution will be saved on the j-th position in SA, while all solutions formerly taking the j-th till $k-1$-th position will drop down one index in the archive and the solution formerly taking the last k-th position is discarded at all. Of course, if the new solution has a worse than attraction as one of s_k, the solution archive remains unaffected. As it is explained in detail in the following, the solutions saved in SA fully imply the deviations and means used for the PDF and imply therefore the major part of the pheromone triplet $(w_l^i, \sigma_l^i, \mu_l^i)$. This way updating the solution archive with better solutions leads automatically a positive pheromone update. Note that a negative pheromone update (evaporation) is indirectly performed as well by the dropping the last solution s_k of SA every time a new solution is introduced in the solution archive. Explicit pheromone evaporation rules are known and can be found for example in Socha and Dorigo [32] but were not considered here due to the implicit negative update and simplicity of the framework.

Standard deviations σ are calculated by exploiting the variety of solutions saved in SA. For every dimension i the maximal and minimal distance between the single solution components s^i of the k solutions saved in SA is calculated. Then the distance between these two values, divided by the number of generations, defines the standard variation for every dimension i:

$$\begin{aligned} \sigma_l^i &= \frac{dis_{\max}(i) - dis_{\min}(i)}{\#\text{generation}} \\ dis_{\max}(i) &= \max\{|s_g^i - s_h^i| :\ g, h \ \in\ \{1, ..., k\},\ g \neq h\} \\ dis_{\min}(i) &= \min\{|s_g^i - s_h^i| :\ g, h \ \in\ \{1, ..., k\},\ g \neq h\} \end{aligned} \tag{5}$$

For all k single Gaussian functions within the PDF this deviation is then used regarding the corresponding dimension i. The means μ are given directly by the single components of the solutions saved in SA:

$$\mu_l^i \ = \ s_l^i \tag{6}$$

The incremental construction of a new ant works the following way: A mean μ_l^i is chosen first for every dimension i . This choice is done respectively to the weights w_l^i. According to the weights defined in (4) and the identity of μ_l^i and s_l^i defined in (6), the mean μ_1^i has the highest probability to be chosen, while μ_k^i has the lowest probability to be chosen. Second, a random number is generated by sampling around the selected mean μ_l^i using the deviation σ_l^i defined by (5). Proceeding like this through all dimensions $i = 1, ..., n$ then creates the new ant, which can be evaluated regarding its objective function value and constraint violations in a next step.

So far, this algorithm framework does not differ from the one proposed by Socha [31], except that here explicit rules for all pheromone parameters $(w_l^i, \sigma_l^i, \mu_l^i)$ are given in (4), (5) and (6), while Socha proposes those only for the deviations and means. It is to note that rules for the deviations and means shown here were taken from Socha. The novel extension which enables the algorithm to handle mixed integer search domains modifies the deviations σ_l^i used for the integer variables and is described now in detail.

To handle integer variables besides the continuous ones, as it is described above, a discretization of the continuous random numbers (sampled by the PDF) is necessary. The clear advantage of this approach is the straight forward integration in the same framework described above. A disadvantage is the missing flexibility in cases where for an integer dimension i all k solution components $s_{1,...,k}^i$ in SA share the same value. In this case the corresponding deviations $\sigma_{1,...,k}^i$ are zero regarding the formulation in (5). As a consequence, no further progress in these components is possible, as the PDF would only sample the exact mean without any deviation.

Introducing a lower bound for the deviation of integer variables helps to overcome this disadvantage and enables the ACO metaheuristic to handle integer and continuous variables simultaneously without any major extension in the same framework. For a dimension i that corresponds to an integer variable the deviations σ_l^i are calculated by:

$$\sigma_l^i \ = \ \max\left\{ \frac{dis_{\max}(i) - dis_{\min}(i)}{\#\text{generation}},\ \ \frac{1}{\#\ \text{generation}},\ (1 - \frac{1}{\sqrt{n_{int}}})/2 \right\} \tag{7}$$

With this definition the deviations according to the corresponding Gaussian functions for integer variables will never fall under a fixed lower bound of $(1 - \frac{1}{\sqrt{n_{int}}})/2$, determined by the number of integer variables n_{int} considered in the MINLP problem formulation. For MINLP with a large amount of integers, this lower bound converges toward 0.5 and ensures therefore a deviation that hopefully keeps the algorithm flexible enough to find its way through the (large) mixed integer search domain. A small number of integer variables leads to a smaller lower bound, whilst for only one integer variable this bound is actually zero. In case of a small amount of integer variables in the MINLP it is reasonable to think that at some point of the search progress the optimal combination of integers is found and therefore no further searching with a wide deviation is necessary for the integer variables. But even in case of only one integer variable, with a corresponding absolute lower bound of zero, the middle term $(\frac{1}{\#\text{generation}})$ in (7) ensures a not too fast convergence of the deviation. Therefore, the calculation of the standard deviation σ for integer variables by (7) seems to be a reasonable choice and is confirmed by the numerical results.

4 Penalty method

Penalty methods are well known strategies to handle constraints in optimization problems. By replacing the original objective function by a penalty function which is a weighted sum (or product) of the original objective function and the constraint violations, the constrained problem can be transformed into an unconstrained one. The penalty function acts therefore as a new objective function which might also be called a cost function. A wide range of modifications of this method is known and comprehensive reviews can be found in Coello [11] or Yeniay [35]. In general it is to note, that simple penalty methods (e.g. *death* or *static*, see Yeniay [35]) do not require a lot of problem specific parameters to be selected, which makes their use and implementation very easy and popular. This advantage of simple penalty methods comes with the drawback, that for especially challenging problems these are often not capable to gain sufficient performance. Sophisticated penalty methods (e.g. *adaptive* or *annealing*, see Yeniay [35]) are more powerful and adjustable to a specific problem due to a larger number of parameters. However, this greater potential implies an additional optimization task for a specific problem: the sufficiently good selection of a large set of parameters. A high potential penalty method, that requires none or only a few parameters to be selected, is therefore an interesting approach.

4.1 Robust oracle penalty method

We present a general penalty method based on only one parameter, named Ω, which is selected best equivalent or just slightly greater than the optimal (feasible) objective function value for a given problem. As for most real-world problems this value is unknown a priori, the user has to guess this parameter Ω at first. After an optimization test run, the quality of the chosen parameter can be directly compared to the truly reached solution value. Due to this predictive nature of the parameter it is called an *oracle*. Despite this illustrative and clear meaning of the oracle parameter, it is important, that the method performs satisfactory even for bad or unreasonable choices of Ω, to apply it to real-world problems. Indeed the method is constructed this way and numerical results fortify the robustness of it. A detailed investigation of the development, properties and robustness of the method is currently in preparation and would go beyond the scope here. Therefore only the essential mathematical formulation together with a brief description, a graphical illustration and a general update rule for the oracle parameter will be given here.

The oracle method works with a Norm-function over all violations of the m constraints of an optimization problem (1); this function is called a residual. Commonly used Norm-functions are the l^1, l^2 or l^∞ Norm, here we assume the l^1 Norm as residual (see (8)). To simplify the notation of the robust oracle method, we denote here with $z := (x, y)$ the vector of all decision variables without explicit respect to x and y, which are the vectors of continuous and integer variables in the MINLP (1) formulation. Such a vector z is called an iterate in the following due to the generality of the method. In case of ACO z represents an ant.

$$res(z) \;=\; \sum_{i=1}^{m_{eq}} |g_i(z)| \;-\; \sum_{i=m_{eq}+1}^{m} \min\{0, g_i(z)\} \tag{8}$$

where $g_{1,\dots,m_{eq}}$ denote the equality constraints and $g_{m_{eq}+1,\dots,m}$ the inequality constraints. The penalty function $p(z)$ is then calculated by:

$$p(z) \;=\; \begin{cases} \alpha \cdot |f(z)-\Omega| + (1-\alpha)\cdot res(z) - \beta & \text{, if} \quad f(z) > \Omega \quad \text{or} \quad res(z) > 0 \\ -|f(z)-\Omega| & \text{, if} \quad f(z) \le \Omega \quad \text{and} \quad res(z) = 0 \end{cases} \tag{9}$$

where α is given by:

$$\alpha \;=\; \begin{cases} \frac{|f(z)-\Omega|\cdot\frac{6\sqrt{3}-2}{6\sqrt{3}} - res(z)}{|f(z)-\Omega| - res(z)} & \text{, if} \quad f(z) > \Omega \quad \text{and} \quad res(z) < \frac{|f(z)-\Omega|}{3} \\ 1 - \frac{1}{2\sqrt{\frac{|f(z)-\Omega|}{res(z)}}} & \text{, if} \quad f(z) > \Omega \quad \text{and} \quad \frac{|f(z)-\Omega|}{3} \le res(z) \le |f(z)-\Omega| \\ \frac{1}{2}\sqrt{\frac{|f(z)-\Omega|}{res(z)}} & \text{, if} \quad f(z) > \Omega \quad \text{and} \quad res(z) > |f(z)-\Omega| \\ 0 & \text{, if} \quad f(z) \le \Omega \end{cases} \tag{10}$$

and β by:

$$\beta \;=\; \begin{cases} \frac{|f(z)-\Omega|\cdot\frac{6\sqrt{3}-2}{6\sqrt{3}}}{1+\frac{1}{\sqrt{\#\text{generation}}}} \cdot \left(1 - \frac{3\;res(z)}{|f(z)-\Omega|}\right) & \text{, if} \quad f(z) > \Omega \quad \text{and} \quad res(z) < \frac{|f(z)-\Omega|}{3} \\ 0 & \text{, else} \end{cases} \tag{11}$$

Before a brief motivation of the single components of the penalty method is given, a graphical illustration of the penalty function values $p(z)$ regarding different objective and residual function values for a given oracle parameter Ω and a fixed number of generations is given below. Figure 1 shows the three dimensional shape of the penalty function according to the first and the hundredth generation with an oracle parameter equal to zero, for residual function values in the range from zero to ten and for objective function values in the range from minus ten to ten. It is important to note, that the shape of the penalty function is not affected at all by the oracle parameter. A lower or greater oracle parameter than zero results only in a movement to the left or right respectively on the axis representing the objective function value.

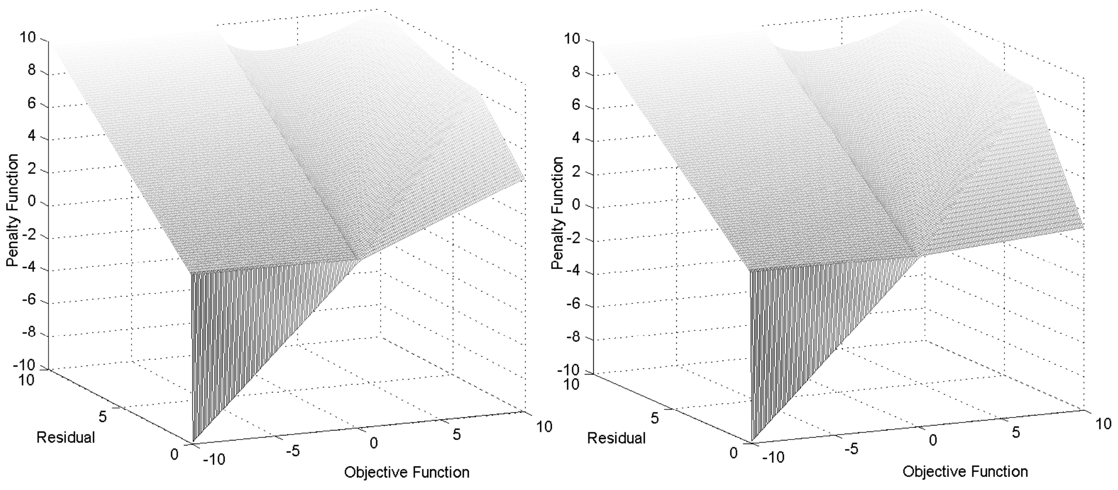

Figure 1: Three dimensional shape of the robust oracle penalty method for $\Omega = 0$ and $\#generation = 1$ (left) and $\#generation = 100$ (right).

Now a brief motivation of the single components of the penalty method is given. The penalty function $p(z)$ defined in (9) is split into two cases. The appearance of the penalty function in case $f(z) > \Omega$ and $res(z) > 0$ is similar to common ones, where a parameter α balances the weight between the objective function and the residual and an additional β term acts as a bias. In case $f(z) \leq \Omega$ and $res(z) = 0$ the penalty function $p(z)$ is defined as the negative distance between the objective function value $f(z)$ and the oracle parameter Ω. In this case, the resulting penalty values will be zero or negative. This case corresponds with the front left sides ($res = 0$ and $f(z) \leq \Omega$) of the 3D shapes shown in Figure 1, which are formed as a vertical triangular. In case $f \leq \Omega$ and $res(z) > 0$ both, the α and β term, are zero. According to (9) this implies, that the penalty value $p(z)$ is equal to the residual value $res(z)$. This case corresponds with the left sides ($f(z) \leq \Omega$) of the two 3D shapes shown in Figure 1, which are formed as a beveled plane.

The two middle terms $1 - \frac{1}{2\sqrt{\frac{|f(z)-\Omega|}{res(z)}}}$ and $\frac{1}{2}\sqrt{\frac{|f(z)-\Omega|}{res(z)}}$ in the definition of α are the major components of the oracle penalty method. In case $f(z) > \Omega$ those are active in α and used respectively to the residual value $res(z)$. If $res(z) > |f(z) - \Omega|$ the latter one is active and results in a value of $\alpha < \frac{1}{2}$, which will increase the weight on the residual in the penalty function $p(z)$. Otherwise, if $res(z) \leq |f(z) - \Omega|$, the first one is active and results in a value of $\alpha \geq \frac{1}{2}$, which will increase the weight on the objective function regarding to (9). Therefore these two components enable the penalty method to balance the search process, respectively to either the objective function value or the residual. This balancing finds it representation in the nonlinear shapes of the upper right sides ($f(z) > \Omega$) of the two 3D shapes shown in Figure 1.

A special case occurs if $f(z) > \Omega$ and $res(z) < \frac{|f(z)-\Omega|}{3}$, in this case the very first term in the definition of α and the β term is active. This is done, because otherwise the second term in the definition of α would lead to a worser penalization of iterates with $res(z) < \frac{|f(z)-\Omega|}{3}$ than those with $res(z) = \frac{|f(z)-\Omega|}{3}$ (a detailed explanation goes beyond the scope here). As this is seen as a negative effect regarding the robustness of the method, because feasible solutions with $res(z) = 0 < \frac{|f(z)-\Omega|}{3}$ would be penalized worse than infeasible ones, the very first term of α is activated in this case. The very first term in α implies an equal penalization $p(z)$ of all iterates sharing the same objective function value $f(z) > \Omega$ and a residual between zero and $\frac{|f(z)-\Omega|}{3}$, not taking the β term in definition (9) into account. The β term goes one step further and biases the penalization of iterates with a smaller residual than $\frac{|f(z)-\Omega|}{3}$ better than those with $res(z) = \frac{|f(z)-\Omega|}{3}$. Moreover this bias will increase with ongoing algorithm process, as the number of generations influence the β term. The front right sides ($f(z) > \Omega$) of the two 3D shapes shown in Figure 1 correspond to this special case, those are formed as a triangular. In case of the left subfigure in Figure 1, which corresponds to a generation number of 1, the biasing effect is not as strong as in case of the right subfigure, which corresponds to a generation number of 100. It is to note, that this dynamic bias, caused by the β term in definition (9), is the only difference between the two 3D shapes shown in Figure 1.

4.2 Oracle update rule

For most real-world problems the global optimal (feasible) objective function value is unknown a priori. As the oracle parameter Ω is selected best, equal or just slightly greater than the optimal objective function value, finding a sufficiently good oracle and solving the original problem to the global optimum goes hand in hand. This self-tuning effect of the method is easy to exploit if several optimization test runs are performed. Each time an optimization test run using a specific oracle has finished, the obtained solution objective and residual function values directly deliver information about an appropriate oracle update.

Numerical tests show that the oracle method is quite robust against overestimated oracles. Underestimated oracles tend to deliver feasible points that are close to but not exactly the global optimal objective function value. Based on those results the following update rule was deduced and is intended to apply, if for a given problem absolutely no information is available and a possible optimal objective function value. The oracle for the very first run should be selected sufficiently low (in Table 1 designated as $-\infty$, e.g. -10^{12}), in this case the robust oracle method follows a *dynamic* penalty strategy. The hope is to find a feasible point, which is already close to the global optimum. If the first run succeeded in finding a feasible solution, the oracle will be updated with this solution and from now on only feasible and lower solutions will be used to further update the oracle. In case the first run did not deliver any feasible point, the oracle should be selected sufficiently large (in Table 1 designated as ∞, e.g. 10^{12}. With this parameter the method will completely focus on finding a feasible solution at first (see the flat shape in Figure 1 for $f(z) < \Omega$ and $res(z) > 0$) and will then switch to a *death* strategy (see the spike shape in Figure 1 for $f(z) < \Omega$ and $res(z) = 0$) as soon as a feasible solution is found.

Table 1: Update rule for the oracle throughout several optimization test runs

Ω^i	Oracle used for the i-th optimization test run
f^i	Obj-function value obtained by the i-th test run
res^i	Residual function value corresponding to f^i
$\Omega^1 = -\infty$ (sufficiently low)	Initialization with *dynamic* strategy
$\Omega^2 = \begin{cases} f^1 & \text{,if } res^1 = 0 \\ \infty \ \text{(sufficiently high)} & \text{,else} \end{cases}$	Update with solution if f^1 is feasible, *static* & *death* strategy if f^1 is infeasible
$\Omega^i = \begin{cases} f^{i-1} & \text{,if } res^{i-1} = 0 \text{ and } f^{i-1} < \Omega^{i-1} \\ \Omega^{i-1} & \text{,else} \end{cases}$	Update Ω^i only with feasible ($res^{i-1} = 0$) and better ($f^{i-1} < \Omega^{i-1} = \{f^{i-2} \vee \infty\}$) solutions

5 Extended ACO implementation

In this section more detailed information about our implementation `ACOmi` is provided. This implementation follows strictly the extended Ant Colony Optimization framework described in Section 3. For unconstrained problems the fitness (or attraction) of an ants is measured by the objective function value while for constrained problems the penalty function value (see Section 4) is used as fitness criteria. Even so this implementation is already capable to handle any kind of MINLP problem; it is additionally hybridized with a mixed integer sequential quadratic programming (MISQP) algorithm by Exler and Schittkowski [19]. This subroutine acts as a local solver within the ACO framework to increase the overall performance of `ACOmi`.

Algorithm 2 gives a pseudo code description of `ACOmi`. A maximal budget of fitness function evaluations or the achievement of a (feasible) solution value lower or equal to a specific objective function value fex is used as stopping criteria. Note, that the value fex is used purely as a stopping criteria and does not correspond to the oracle parameter Ω within the algorithm. The extended ACO algorithm requires two parameters to be selected (n_{pop} and k). For constrained problems the oracle parameter Ω must be provided to apply the penalty method.

To investigate the performance of `ACOmi` with and without the local solver, the use of MISQP is designed in `ACOmi` as an optional choice. Furthermore two different hybridization options (option1 and option2) are considered. In case option1 is activated, the local solver will be called after every generation, using the currently best found solution as initial point. In case option2 is activated, the local solver will be called only one time after the last but one generation, using the currently best found solution as initial point. This is done after the last but one generation, so to not exceed the maximal number of function evaluations. Even though small violations of this budget are possible, the number of function evaluations required by MISQP can not be known in advance.

Algorithm 2 `ACOmi`

```
select maximal number of function evaluations: $eval_{max}$
select objective function value to be reached: $fex$
----------------------------------------------------------------
select population size: $n_{pop}$
select number of PDF kernels: $k$
select oracle: $\Omega$ (for constrained problems)
----------------------------------------------------------------
option1: run local solver after every generation (Yes/No)
option2: run local solver after the last but one generation (Yes/No)
----------------------------------------------------------------
initialize solution archive $SA$ of size $k$ (empty)
initialize pheromones (randomly)
----------------------------------------------------------------
while stopping criteria not met do
  for $i$ from 1 to $n_{pop}$ do
    construct ant $a_i$ regarding pheromones
    evaluate fitness of ant $a_i$
    update solution archive with ant $a_i$
  end for
  update pheromones according to $SA$
  if option1 then
    run local solver MISQP
  end if
  if option2 then
    run local solver MISQP
  end if
end while
```

Note, that it is possible to submit a starting solution to `ACOmi` as it is done for the numerical results in Section 6. This solution will be treated like any other ant during the first generation of ants, which are created according to the initially (randomly) selected pheromones. Based on the comparison of the fitness function value of the starting solution and the fitness of the other ants, it will be introduced in the solution archive SA or not.

6 Numerical Results

We evaluate the performance of our implementation ACOmi using (i) a set of 26 MINLP benchmark problems, and (ii) an engineering design problem arising from a thermal insulation system. Regarding the parameter setting of ACOmi Table 2 presents the default values which are based only on experimental experience. Those were applied on all numerical test runs of ACOmi presented in this paper. All test problems discussed in this paper are constrained, therefore the penalty function described in Section 4 acts as fitness criteria in ACOmi (see Section 5). As several test runs are performed on the problems, the oracle update rule (see Table 1) is applied.

Table 2: Default parameter setting of ACOmi

parameter	value	explanation	reference
n_{pop}	150	number of ants	Section 5
k	15	number of kernels = size of SA	Section 3
Ω	*updated*	oracle parameter for penalty function	Table 1

To compare the performance of our implementation, a mixed integer tabu-search implementation (MITS) by Exler et al. [18] and a mixed integer sequential programming (MISQP) implementation by Exler and Schittkowski [19] are considered as well. It is to note that the latter is a deterministic local solver which both implementations, MITS and ACOmi, are hybridized with. Furthermore MISQP as a local solver is not intended for the global optimization of non-convex optimization problems, like the ones considered in the following. To still be able to compare the performance of MISQP, a fitness function evaluation limited multistart of MISQP with random initial points is considered here as concurrent method to MITS and ACOmi.

Of special interest to the reader might be the performance of ACOmi with and without the use of the local solver. Therefore three different ACOmi setups regarding the use of MISQP are considered for the numerical tests, those setups are described in Table 3. The first setup (ACOmi$_1$) is the pure ACO algorithm without any local solver activity. The second setup (ACOmi$_2$) calls MISQP after every generation which implies a massive exploitation of the hybridization. The third setup (ACOmi$_3$) calls the local solver only one time at the very end of the search process, when the maximal limit of function evaluations is nearly reached (after the one but last ACO generation). MISQP is always started with the currently best solution found by ACOmi as starting point. An additional setup of MITS without the local solver would have been interesting to compare the results with those of ACOmi$_1$. Unfortunately the local solver is not an optional choice in MITS and can not be deactivated.

Table 3: Different ACOmi setups regarding the local solver

setup	option	activated	explanation
ACOmi$_1$:	option1	No	run MISQP after every generation
	option2	No	run MISQP after the last but one generation
ACOmi$_2$:	option1	Yes	run MISQP after every generation
	option2	No	run MISQP after the last but one generation
ACOmi$_3$:	option1	No	run MISQP after every generation
	option2	Yes	run MISQP after the last but one generation

6.1 MINLP Benchmark problems

A set of 26 MINLPs benchmark problems from the open literature is considered in this section. As this paper is focused on global optimization of non-convex problems in general, we selected problems with a high level of non-convexity in order to check if our method was able to deal with the likely multimodal search domain. For each problem on this selection a frequency histogram, obtained by multistart executions of a local solver from random initial points (see Section 6.2 for an example), was performed. The profiles of these histograms illustrated the multimodality of each problem. Problems with a highly multimodal search domain are marked with an * in Table 5 and are therefore considered more difficult to solve. Problems without an * are either convex (thus unimodal), or contain only a small number of local optima. Besides the specific problem name, Table 5 also lists the number and type of decision variables and constraints, and the best known objective function value for every problem. Table 4 explains the used abbreviations in Table 5 in detail.

Table 4: Abbreviations for Table 5

abbreviation	explanation
name	problem name used in the literature
ref	literature reference
n_{con}	number of continuous variables
n_{int}	number of integer variables (without binary ones)
n_{bin}	number of binary variables
m_{eq}	number of equality constraints
m	number of constraints in total
fex	best known objective function value

Table 5: MINLP Benchmark problems

name	ref	n_{con}	n_{int}	n_{bin}	m_{eq}	m	fex
mitp1	[19]	2	3	0	0	1	-10009.69
mitp3	[19]	2	0	3	0	7	3.5
van de braak 1	[19]	4	3	0	0	2	1
van de braak 2*	[19]	4	3	0	0	4	-2.718281
van de braak 3*	[19]	4	3	0	0	4	-89.8
gear	[19]	4	4	0	4	4	1
asaadi 1.1	[3]	1	3	0	0	3	-40.956609
asaadi 2.1	[3]	3	4	0	0	4	694.903
asaadi 3.1	[3]	4	6	0	0	8	37.21954
tp83	[24]	3	2	0	0	6	-30665.5386
wp02	[34]	1	1	0	0	2	-2.444
nvs08*	[23]	1	2	0	0	3	23.44973
nvs14	[23]	3	5	0	3	3	-40358.15
nvs20*	[23]	11	5	0	0	8	195.310251
duran/grossmann 1	[16]	3	0	3	0	6	6.0097418
duran/grossmann 2	[16]	6	0	5	1	14	0.73035665
duran/grossmann 3	[16]	9	0	8	2	23	68.01
floudas 1	[20]	2	0	3	2	5	7.6671801
floudas 2	[20]	2	0	1	0	3	1.07654
floudas 3*	[20]	3	0	4	0	9	4.5795825
floudas 4*	[20]	3	0	8	3	7	-0.96156973
floudas 6*	[20]	1	1	0	0	3	-17
floudas/pardalos 3.3*	[20]	3	3	0	0	6	-310
windfac*	[28]	11	3	0	13	13	0.25449
batch*	[28]	22	0	24	0	81	285507
optprloc	[28]	5	0	25	0	29	-8.0641362

The results obtained by MITS, the MISQP multistart and the ACOmi setups for all the problems are listed in Table 7. All problems were solved 30 times each with a maximal budget of 10000 fitness function evaluations. Besides the maximal evaluation budget $eval_{max}$ the achievement of the best known objective function value fex was applied as stopping criteria in `ACOmi`, MITS and the MISQP multistart. A moderate precision of 10^{-4} regarding the l^1 Norm of all constraint violations and the objective function value was claimed. Even so higher precisions can be easily obtained by the deterministic MISQP routine, the `ACOmi` setup `ACOmi`$_1$ without any local solver is a pure stochastic algorithm, which might require a large amount of function evaluations to achieve solutions with higher precision.

For every test run of every problem a random point was created and submitted to all solvers as starting point. This procedure ensures a fair competition between the MITS, the MISQP multistart and the `ACOmi` setups. All results for the MINLP benchmark problems can be found in Table 7, Table 6 explains the abbreviations used in Table 7.

Table 6: Abbreviations for Table 7

abbreviation	explanation
name	problem name used in the literature
solver	solver corresponding to the line of results
feasible	number of feasible solutions found out of 30 test runs
optimal	number of optimal solutions found out of 30 test runs
f_{best}	best (feasible) objective function value found out of 30 test runs
f_{worst}	worst (feasible) objective function value found out of 30 test runs
f_{mean}	mean objective function value over all runs with a feasible solution
time$_{\text{mean}}$	mean cpu-time over all runs with a feasible solution
eval$_{\text{mean}}$	mean number of evaluations over all runs with a feasible solution

Table 7: Results for the MINLP Benchmark problems

name	solver	feasible	optimal	f_{best}	f_{worst}	f_{mean}	eval$_{mean}$	time$_{mean}$
mitp1	`ACOmi`$_1$	30	30	-10010	-10010	-10010	1641	0.45
	`ACOmi`$_2$	30	30	-10010	-10010	-10010	525	0.11
	`ACOmi`$_3$	30	30	-10010	-10010	-10010	1656	0.44
	MISQP	30	30	-10010	-10010	-10010	330	0.05
	MITS	30	30	-10010	-10010	-10010	296	0.08
mitp3	`ACOmi`$_1$	30	27	3.4998	4.9997	3.6023	3196	0.77
	`ACOmi`$_2$	30	30	3.5	3.5	3.5	345	0.08
	`ACOmi`$_3$	30	30	3.4998	3.5001	3.5	3145	0.76
	MISQP	30	30	3.5	3.5	3.5	43	0.01
	MITS	30	30	3.5	3.5	3.5	44	0.02
van de braak 1	`ACOmi`$_1$	30	29	1	1.0092	1.0004	4041	1.13
	`ACOmi`$_2$	30	30	1	1.0001	1	1008	0.17
	`ACOmi`$_3$	30	30	1	1.0001	1.0001	4268	1.18
	MISQP	30	30	1	1	1	379	0.05
	MITS	30	30	1	1.0001	1	779	0.18
van de braak 2	`ACOmi`$_1$	30	0	9517.3	2.0078e+006	8.4065e+005	10051	2.67
	`ACOmi`$_2$	30	1	-2.7183	2.2778e+006	7.5726e+005	9775	2.47
	`ACOmi`$_3$	30	1	-2.7182	1.9441e+006	6.7533e+005	9947	2.58
	MISQP	30	29	-2.7183	-1	-2.661	2827	0.74
	MITS	30	30	-2.7183	-2.7182	-2.7183	813	0.31
van de braak 3	`ACOmi`$_1$	30	21	-89.8	42.147	-85.213	4816	1.34
	`ACOmi`$_2$	30	29	-89.8	-41.639	-88.195	1781	0.39
	`ACOmi`$_3$	30	29	-89.8	-47.599	-88.393	4289	1.24
	MISQP	30	30	-89.8	-89.8	-89.8	1449	0.20
	MITS	30	26	-89.8	-89.799	-89.8	996	0.23
gear	`ACOmi`$_1$	29	0	1.0696	2.5203	1.7532	10051	2.92
	`ACOmi`$_2$	30	30	1	1.0001	1	506	0.14
	`ACOmi`$_3$	30	30	1	1.0001	1	10057	2.92
	MISQP	30	30	1	1	1	214	0.04
	MITS	30	30	1	1.0001	1	155	0.05
asaadi 1.1	`ACOmi`$_1$	30	30	-40.958	-40.957	-40.957	1371	0.34
	`ACOmi`$_2$	30	30	-40.958	-40.957	-40.957	389	0.08
	`ACOmi`$_3$	30	30	-40.958	-40.957	-40.957	1301	0.32
	MISQP	30	30	-40.958	-40.958	-40.958	98	0.02
	MITS	30	30	-40.958	-40.957	-40.957	91	0.03

(continued)

name	solver	feasible	optimal	f_{best}	f_{worst}	f_{mean}	eval$_{mean}$	time$_{mean}$
asaadi 2.1	ACOmi$_1$	30	30	694.9	694.9	694.9	1626	0.46
	ACOmi$_2$	30	30	694.9	694.9	694.9	615	0.13
	ACOmi$_3$	30	30	694.9	694.9	694.9	1681	0.47
	MISQP	30	30	694.9	694.9	694.9	501	0.08
	MITS	30	30	694.9	694.9	694.9	314	0.11
asaadi 3.1	ACOmi$_1$	30	24	37.219	40.534	37.854	5306	1.67
	ACOmi$_2$	30	30	37.219	37.219	37.219	762	0.22
	ACOmi$_3$	30	28	37.219	40.347	37.428	4717	1.49
	MISQP	30	30	37.219	37.219	37.219	333	0.08
	MITS	30	30	37.219	37.219	37.219	317	0.11
tp83	ACOmi$_1$	30	27	-30666	-29924	-30636	3371	0.86
	ACOmi$_2$	30	30	-30666	-30666	-30666	346	0.08
	ACOmi$_3$	30	30	-30666	-30666	-30666	2807	0.71
	MISQP	30	30	-30666	-30666	-30666	62	0.01
	MITS	30	30	-30666	-30666	-30666	62	0.02
wp02	ACOmi$_1$	30	30	-2.4444	-2.4443	-2.4444	256	0.06
	ACOmi$_2$	30	30	-2.4444	-2.4443	-2.4444	270	0.06
	ACOmi$_3$	30	30	-2.4444	-2.4444	-2.4444	291	0.07
	MISQP	30	30	-2.4444	-2.4444	-2.4444	43	0.01
	MITS	30	30	-2.4444	-2.4444	-2.4444	63	0.02
nvs08	ACOmi$_1$	30	30	23.45	23.45	23.45	2051	0.49
	ACOmi$_2$	30	28	23.45	23.828	23.475	1180	0.25
	ACOmi$_3$	30	28	23.45	23.828	23.475	2554	0.60
	MISQP	30	30	23.45	23.45	23.45	161	0.02
	MITS	30	27	23.45	23.45	23.45	226	0.06
nvs14	ACOmi$_1$	14	2	-40358	-31672	-38636	9644	2.80
	ACOmi$_2$	30	30	-40358	-40358	-40358	689	0.18
	ACOmi$_3$	30	29	-40358	-40256	-40355	9194	2.46
	MISQP	30	30	-40358	-40358	-40358	694	0.13
	MITS	30	30	-40358	-40358	-40358	352	0.11
nvs20	ACOmi$_1$	30	0	196.48	3106.9	950.75	10051	3.54
	ACOmi$_2$	30	30	195.31	195.31	195.31	4489	1.08
	ACOmi$_3$	30	28	195.31	195.78	195.33	11276	3.88
	MISQP	30	30	195.31	195.31	195.31	1124	0.22
	MITS	30	30	195.31	195.31	195.31	2380	0.74
duran/ grossmann 1	ACOmi$_1$	30	28	6.0084	9.1796	6.1157	2971	0.83
	ACOmi$_2$	30	29	6.0094	7.0925	6.0457	990	0.23
	ACOmi$_3$	30	27	6.0084	7.0926	6.1175	3240	0.90
	MISQP	30	30	6.0095	6.0098	6.0097	507	0.08
	MITS	30	30	6.0094	6.0098	6.0097	173	0.05
duran/ grossmann 2	ACOmi$_1$	7	0	75.043	99.208	83.685	10051	3.45
	ACOmi$_2$	23	23	73.035	73.035	73.035	1069	0.39
	ACOmi$_3$	11	11	73.035	73.035	73.035	10195	3.38
	MISQP	30	30	73.035	73.035	73.035	453	0.10
	MITS	29	29	73.034	73.034	73.034	572	0.18
duran/ grossmann 3	ACOmi$_1$	6	0	78.109	88.669	82.931	10051	3.89
	ACOmi$_2$	30	30	68.009	68.01	68.01	1262	0.78
	ACOmi$_3$	15	12	68.01	77.104	69.397	10482	4.16
	MISQP	30	30	68.01	68.01	68.01	515	0.20
	MITS	30	30	68.008	68.008	68.008	616	0.50
floudas 1	ACOmi$_1$	0	0	na	na	na	na	na
	ACOmi$_2$	30	30	7.6672	7.6672	7.6672	363	0.09
	ACOmi$_3$	30	30	7.6672	7.6672	7.6672	9963	2.86
	MISQP	30	30	7.6672	7.6672	7.6672	235	0.04
	MITS	30	30	7.6672	7.6672	7.6672	248	0.07

(continued)

name	solver	feasible	optimal	f_{best}	f_{worst}	f_{mean}	eval_{mean}	time_{mean}
floudas 2	$\texttt{ACOmi}_1$	30	0	1.103	1.35	1.2483	10051	2.62
	$\texttt{ACOmi}_2$	30	18	1.0765	1.25	1.1459	4250	1.10
	$\texttt{ACOmi}_3$	30	10	1.0765	1.25	1.1922	9822	2.50
	MISQP	30	30	1.0765	1.0765	1.0765	53	0.01
	MITS	30	30	1.0765	1.0766	1.0765	59	0.02
floudas 3	$\texttt{ACOmi}_1$	30	22	4.5793	5.8138	4.8929	5976	1.79
	$\texttt{ACOmi}_2$	30	30	4.5796	4.5796	4.5796	731	0.99
	$\texttt{ACOmi}_3$	30	30	4.5794	4.5797	4.5795	6999	2.06
	MISQP	30	30	4.5795	4.5796	4.5796	1617	6.12
	MITS	30	29	4.5795	6.5574	4.6455	2864	5.05
floudas 4	$\texttt{ACOmi}_1$	28	0	-0.7641	-5.4612e-005	-0.051609	10051	3.24
	$\texttt{ACOmi}_2$	30	1	-0.96157	-0.9532	-0.95348	9813	2.91
	$\texttt{ACOmi}_3$	30	0	-0.9532	-3.3434e-006	-0.91512	10227	3.34
	MISQP	30	19	-0.96157	-0.9532	-0.9585	6513	2.21
	MITS	30	30	-0.96157	-0.96157	-0.96157	1173	0.48
floudas 6	$\texttt{ACOmi}_1$	30	30	-17	-17	-17	956	0.24
	$\texttt{ACOmi}_2$	30	30	-17	-17	-17	307	0.08
	$\texttt{ACOmi}_3$	30	30	-17	-17	-17	981	0.25
	MISQP	30	30	-17	-17	-17	9	0.00
	MITS	30	30	-17	-17	-17	31	0.01
floudas/	$\texttt{ACOmi}_1$	30	7	-310.01	-121.93	-272.77	8356	2.31
pardalos	$\texttt{ACOmi}_2$	30	26	-310	-298	-308.4	2219	0.58
3.3	$\texttt{ACOmi}_3$	30	25	-310.01	-298	-308	6656	1.82
	MISQP	30	30	-310	-310	-310	440	0.06
	MITS	30	30	-310	-310	-310	1321	0.33
windfac	$\texttt{ACOmi}_1$	0	0	na	na	na	na	na
	$\texttt{ACOmi}_2$	30	24	0.25448	0.42881	0.28736	4037	1.08
	$\texttt{ACOmi}_3$	29	14	0.25441	0.75	0.34117	10203	3.44
	MISQP	29	17	0.25448	0.75	0.32523	6894	2.70
	MITS	30	18	0.25439	0.74985	0.31928	7241	4.10
batch	$\texttt{ACOmi}_1$	0	0	na	na	na	na	na
	$\texttt{ACOmi}_2$	25	21	2.8551e+005	3.4673e+005	2.9019e+005	5333	24.89
	$\texttt{ACOmi}_3$	8	3	2.8551e+005	3.0569e+005	2.9269e+005	11457	33.42
	MISQP	21	17	2.8551e+005	3.8248e+005	2.9033e+005	7124	23.00
	MITS	29	29	2.8551e+005	2.8551e+005	2.8551e+005	5367	16.81
optprloc	$\texttt{ACOmi}_1$	10	0	-5.9625	0.37749	-2.9912	10051	19.48
	$\texttt{ACOmi}_2$	30	30	-8.0641	-8.0641	-8.0641	1176	4.47
	$\texttt{ACOmi}_3$	30	29	-8.0641	-7.9152	-8.0592	10713	22.28
	MISQP	30	30	-8.0641	-8.0641	-8.0641	1123	4.50
	MITS	30	30	-8.0641	-8.0641	-8.0641	850	3.06

To illustrate the results achieved by MITS, the MISQP multistart and the `ACOmi` setups in a very compact way, we present a performance profile based on the number of fitness function evaluations. Following Auger and Hansen [4] we define the success performance FE for a solver on a specific problem by:

$$\mathsf{FE} \;=\; \text{eval}_{mean} \cdot \frac{\#\text{all runs(30)}}{\#\text{successful runs}} \tag{12}$$

Where eval_{mean} is the mean number of function evaluations of all successful runs. As a successful run only those are considered, that obtained the best known global solution. With this definition the best success performance FEbest is given for every problem by the lowest value of FE respectively to all considered solvers. Figure 2 shows the empirical distribution function of the success performance FE divided by FEbest on the respective test problem. Every line represents the performance of a specific solver, every step indicates that a problem was solved to the global optimal solution at least once out of the 30 test runs.

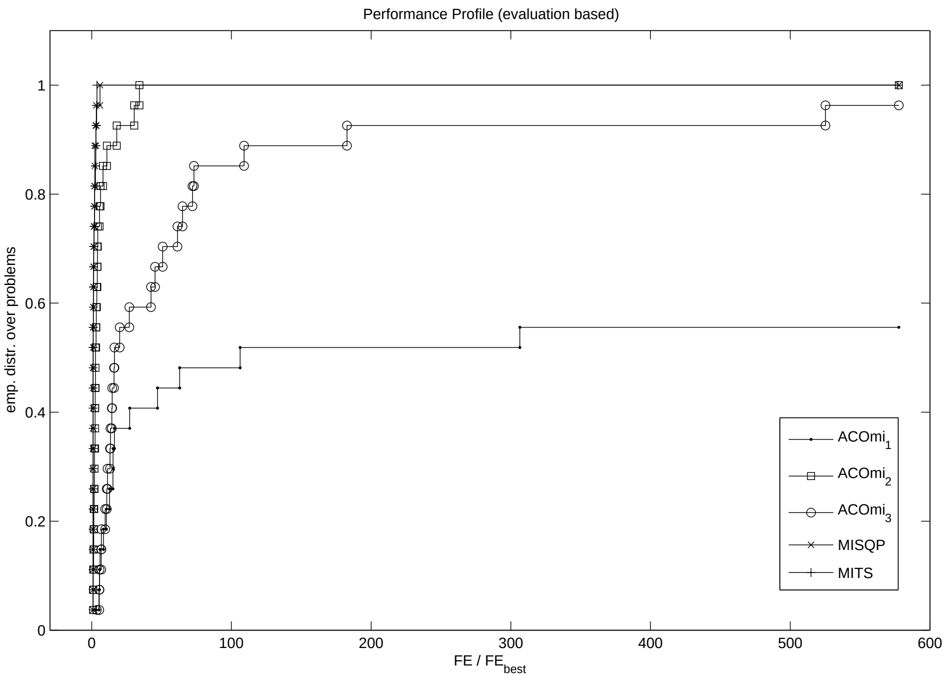

Figure 2: Performance profile for MITS, the MISQP multistart and the `ACOmi` setups for the MINLP benchmark problems

Note that the performance profile shown in Figure 2 is a drastic simplification of the detailed results shown in Table 7, as only test runs that delivered the global optimal solution have an impact on the success performance FE. Clearly the best solver for this set of benchmark problems is MITS, directly followed by the MISQP multistart. Both were able to deliver global optimal solution for every problem of the set, but MITS was more robust and slightly faster in most cases. The fact, that the MISQP multistart was capable to achieve global optimal solutions in all cases within a highly competitive number of function evaluations in comparison to the global solver MITS, proves the strength of this local solver. As MISQP is essentially embedded in MITS, the results obtained by MITS on this set must be seen under the aspect of this hybridization.

Taking into account the strength of MISQP on this set of problems, it is not very surprising, that the `ACOmi` setup exploiting the local solver most was the best one among the three setups. $\texttt{ACOmi}_2$ was able to obtain global optimal solutions for every problem. Except for two problems (van de braak 2, floudas 4) the performance of $\texttt{ACOmi}_2$ according to the robustness and speed are even highly competitive to those of MITS or the MISQP multistart. Obviously the worst `ACOmi` setup for this set of problems was $\texttt{ACOmi}_1$ which did not benefit of the local solver at all. As this setup represents the here proposed extended ACO framework and oracle penalty method, a closer look on the results is necessary. Even so $\texttt{ACOmi}_1$ was able to find the global optimal solution in only 58% of the problems, only for three problems (floudas 1, windfac, batch) no feasible solution could be found at all. Among the problems $\texttt{ACOmi}_1$ succeeded in finding feasible solutions but not global ones, the solutions are often still good (e.g. gear, nvs20, duran/grossmann 2, floudas 2, floudas 4).

Assuming the third `ACOmi` setup $\texttt{ACOmi}_3$, performing a pure ACO search and calling MISQP only one time at the very end, the quality of the solutions provided by $\texttt{ACOmi}_1$ become more evident. Except for one problem (floudas 4) the pure ACO search provided solutions that were close enough to the global optimum, that the local solver succeeded in obtaining the global solution. In other words, even if the pure ACO search by $\texttt{ACOmi}_1$ did not succeed in achieving the global solution with the claimed precision in the given budget of function evaluations, it acted as a sufficiently good starting point for the local solver to obtain the global optimal solution.

6.2 Thermal insulation system application

In this section we describe the application of `ACOmi` on a load-bearing thermal insulation system. The corresponding model files ('Heat Shield Problem Files') to this application were obtained from Abramson [2]. The 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. 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 with 100 random initial points is shown in Figure 6.2.1. 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.

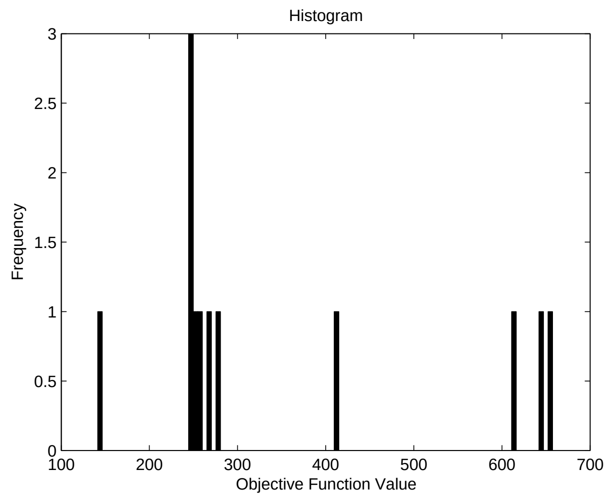

Figure 3: Frequency histogram of feasible solutions for the 'Heat Shield Problem' found by a multistart of MISQP with 100 random initial points (only solutions with an objective function value lower than 1000 are displayed)

Once again we compare the performance of the three `ACOmi` setups (see Table 3) with the one of MITS and an evaluation limited multistart of MISQP with random initial points, like in Section 6.1. The default settings of `ACOmi` (see Table 2) are used. A maximal budget of 10000 function evaluations was set for every solver and 30 test runs have been performed. Again a precision of 10^{-4} regarding the l^1 Norm of all constraint violations and the objective function value was claimed. As the Heatshield problem contains two inequality constraints, `ACOmi` uses the robust oracle penalty method to evaluate the fitness of ants. The oracle parameter is chosen accordingly to the update rule described in Section 4.2 throughout the test runs.

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. [26]. 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 8, 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 evaluations ($\text{eval}_{\text{mean}}$) and the corresponding cpu-time ($\text{time}_{\text{mean}}$) in seconds is also reported for all solvers.

Table 8: Results for the Heatshield problem

solver	feasible	f_{best}	f_{worst}	f_{mean}	eval$_{mean}$	time$_{mean}$
`ACOmi`$_1$	30	105.86	110.01	107	10051	206.03
`ACOmi`$_2$	30	105.79	163.54	112.9	10370	331.54
`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 8, Table 9 gives detailed information on the best decision vectors x^* obtained by MITS and `ACOmi` (best solution obtained by setup `ACOmi`$_2$). Also the above mentioned NOMADm solution and the solution values (original and normalized) are given.

Table 9: Best solution x^* by NOMADm, MITS and `ACOmi`

solution information	x^*_{NOMADm}	x^*_{MITS}	$x^*_{\texttt{ACOmi}_2}$
continuous variables:			
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:			
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 $\texttt{ACOmi}_1$ setup are given in Figure 4. Note that the plots uses double logarithmic scale.

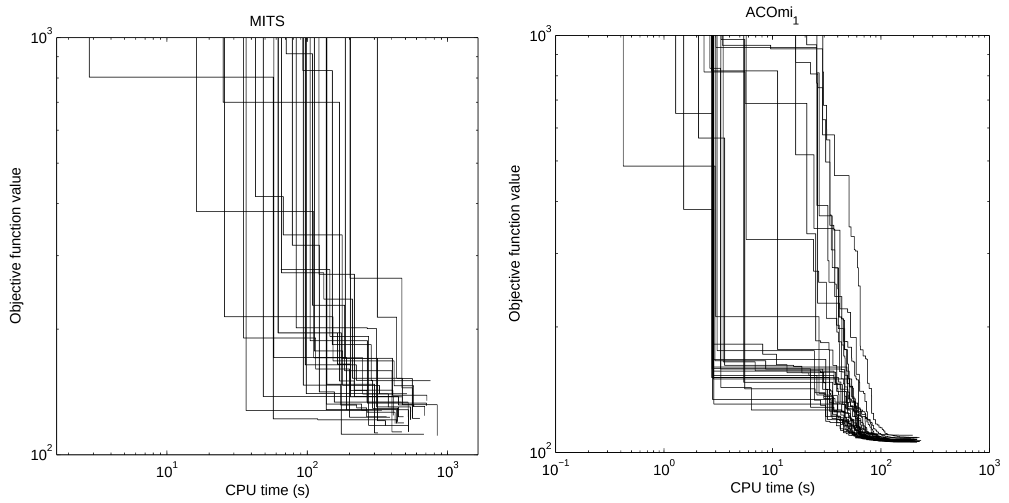

Figure 4: All 30 convergence curves of MISQP and ACOmi$_1$ for the Heatschield problem

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. Bearing this in mind, the contrary performance of the tested solvers on this application and the previous MINLP benchmark set becomes understandable.

7 Conclusions

We presented an extension of the Ant Colony Optimization metaheuristic enabling the methodology to handle mixed integer variable search domains. Furthermore we introduced a new penalization strategy which is applied in our extended ACO implementation ACOmi. Numerical tests of ACOmi on a set of 26 MINLP benchmark problems and one engineering design problem have been performed with a particular focus on the benefit of a local solver used within ACOmi. It turned out that on the set of MINLP benchmark problems, where a multistart of the local solver MISQP was very successful, the massive use of the hybridized local solver was most promising in ACOmi and delivered inferior but competitive results to the concurrent MITS solver. Nevertheless the pure ACO search was capable to deliver good solutions in most of the cases on this challenging benchmark problems.

The results obtained by ACOmi on the engineering design problem were outperforming those of MITS, the MISQP multistart and even the two reported solutions in Abramson [1]. Furthermore the pure ACO search in ACOmi was responsible for the robust success of ACOmi on this application.

Acknowledgments

We acknowledge the support of the EU and the Marie-Curie Actions program. We thank the project PRISM - MRTN-CT-2004-512233 - *Towards Knowledge-based Processing Systems* for the support. Jose A. Egea gratefully acknowledges financial support (FPU fellowship) from the Spanish Ministry of Education and Science.

References

[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. Http://www.afit.edu/en/ENC/Faculty /MAbramson/NOMADm.html, 2007.

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

[4] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, 2005.

[5] C. Blum. The metaheuristics network web site: Ant colony optimization. Http://www.metaheuristics.org/index.php?main=3&sub=31, 2008.

[6] F. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature*, 406:39–42, 2000.

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

[8] R. Chelouah and P. Siarry. Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *Eur. J. Oper. Res.*, 148(2):335–348, 2003.

[9] R. Chelouah and P. Siarry. A hybrid method combining continuous tabu search and nelder-mead simplex algorithms for the global optimization of multiminima functions. *Eur. J. Oper. Res.*, 161(3):636–654, 2005.

[10] W. Chunfeng and Z. Xin. Ants foraging mechanism in the design of multiproduct batch chemical process. *Ind. Eng. Chem. Res.*, 41(26):6678–6686, 2002.

[11] C. A. 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.

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

[13] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artifical Life*, 5(2):137–172, 1999.

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

[15] J. Dreo and P. Siarry. An ant colony algorithm aimed at dynamic continuous optimization. *Appl. Math. Comput.*, 181(1):457–467, 2006.

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

[17] J. A. Egea, M. Rodriguez-Fernandez, J. R. Banga, and R. Marti. Scatter search for chemical and bio-process optimization. *J. Global. Optim.*, 37(3):481–503, 2007.

[18] 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.*, 2008, in press.

[19] O. Exler and K. Schittkowksi. A trust region sqp algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 3(1):269–280, 2007.

[20] C. Floudas, P. Pardalos, C. Adjiman, W. Esposito, G. Z.H., S. Harding, J. Klepeis, C. Meyer, and C. Schweiger. *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers, 1999.

[21] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics.* Springer, New York, 2003.

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

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

[24] S. K. Hock W. Test examples for nonlinear programming codes. *Lect. Notes Econ. Math.*, 187:127–129, 1981.

[25] V. K. Jayaraman, B. D. Kulkarni, S. Karale, and P. Shelokar. Ant colony framework for optimal design and scheduling of batch plants. *Comput. Chem. Eng.*, 24(8):1901–1912, 2000.

[26] D. J. 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.

[27] M. Kong and P. Tian. *Application of ACO in Continuous Domain*, pages 126–135. Advances in Natural Computation. Springer, Berlin, Heidelberg, 2006.

[28] S. Leyffer. Macminlp, ampl collection of mixed integer nonlinear programs. Http://www-unix.mcs.anl.gov/ leyffer/macminlp/index.html, 2000.

[29] J. Rajesh, K. Gupta, H. S. Kusumakar, V. K. Jayaraman, and B. D. Kulkarni. Dynamic optimization of chemical processes using ant colony framework. *Comput. Chem.*, 25(6):583–595, 2001.

[30] A. T. Serban and G. Sandou. *Mixed Ant Colony Optimization for the Unit Commitment Problem*, pages 332–340. Adaptive and Natural Computing Algorithms. Springer, Berlin, Heidelberg, 2007.

[31] K. Socha. *ACO for Continuous and Mixed-Variable Optimization*, pages 25–36. Ant Colony, Optimization and Swarm Intelligence. Springer, Berlin, Heidelberg, 2004.

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

[33] T. Stuetzle and M. Dorigo. *ACO algorithms for the travelling salesman problem*, pages 163–183. Evolutionary Algorithms in Engineering and Computer Science. John Wiley & Sons, Chichester (UK), 1999.

[34] T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optim. Eng.*, 3:253–280, 2002.

[35] O. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10:45–56, 2005.

[36] L. Yu, K. Liu, and K. Li. Ant colony optimization in continuous problem. *Frontiers of Mechanical Engineering in China*, 2(4):459–462, 2007.

[37] B. Zhang, D. Chen, and W. Zhao. Iterative ant-colony algorithm and its application to dynamic optimization of chemical process. *Comput. Chem. Eng.*, 29(10):2078–2086, 2005.