# A Numerical Study of MIDACO on 100 MINLP Benchmarks

Martin Schlüter*, Matthias Gerdts†, Jan-J. Rückmann*

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

† *Institut fuer Mathematik und Rechneranwendung, Universitaet der Bundeswehr Muenchen,*
*D-85577 Neubiberg/Muenchen, Germany*
{matthias.gerdts@unibw.de}

January 31, 2012

**Abstract**

This paper presents a numerical study on MIDACO, a new global optimization software for mixed integer nonlinear programming (MINLP) based on ant colony optimization and the oracle penalty method. Extensive and rigorous numerical tests on a set of 100 non-convex MINLP benchmark problems from the open literature are performed. Results obtained by MIDACO are directly compared to results by a recent study of state of the art deterministic MINLP software on the same test set. Further comparisons with established MINLP software is undertaken in addition. This study shows, that MIDACO is not only competitive to established MINLP software, but can even outperform those in terms of number of global optimal solutions found. Moreover, the parallelization capabilities of MIDACO enable it to be even competitive to deterministic software regarding the amount of (serial processed) function evaluation, while the black-box capabilities of MIDACO offer an intriguing new robustness for MINLP.

**Keywords: Mixed Integer Nonlinear Programming (MINLP), Ant Colony Optimization, Oracle Penalty Method, Parallel Computing, MIDACO, MISQP, BONMIN, COUENNE.**

## 1 Introduction

Mixed integer nonlinear programming (MINLP) problems are an important class of optimization problems with many real world applications. A mathematical formulation of a MINLP is given in (1), where $f(x, y)$ denotes the objective function to be minimized. In (1), the equality constraints are given by $g_{1,\ldots,me}(x, y)$ and the inequality constraints are given by $g_{me+1,\ldots,m}(x, y)$. The vector $x$ contains the continuous decision variables and the vector $y$ contains the discrete decision variables. Furthermore, some box constraints as $x_l, y_l$ (lower bounds) and $x_u, y_u$ (upper bounds) for the decision variables $x$ and $y$ are considered in (1).

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

MINLP problems are known to be difficult to solve. This is especially true, if the objective or constraint functions are non-convex. Several deterministic approaches are well known and established to solve MINLP problems. The most common ones are Branch and Bound, Outer Approximation, Generalized Benders Decomposition, Extended Cutting Plane and Sequential Quadratic Programming (SQP) based methods. A review on deterministic MINLP algorithms can be found in Grossmann [12]. A recent and comprehensive overview on MINLP software is presented in Bussieck and Vigerske [3].

Contrary to those deterministic algorithms there are only few stochastic approaches on MINLP. The most common stochastic approach is supposedly OQNLP [24], which is a hybrid algorithm, combining a stochastic framework with a deterministic local solver. In terms of purely stochastic approaches (this means without any combination with a deterministic method), there is currently no algorithm known, that has been tested and compared on a meaningful set of MINLP problems.

Stochastic optimization algorithms are conceptually very different from deterministic ones. While deterministic algorithms often come with a profound theoretical analysis, stochastic ones mostly lack of this (due to the difficult examination of their stochastic behavior). In addition to this, stochastic algorithms normally require much more function evaluation than their deterministic counterparts. On the other hand, stochastic algorithms can offer the intriguing advantage of black-box optimization. This means, that the objective and constraint functions and their properties can be completely unknown and even exhibit critical properties like non-convexity, discontinuities, flat spots or stochastic distortions. As MINLP problems often model complex real world applications, which are likely to include those properties (see [19] or [20] for examples), stochastic approaches can offer an interesting advantage over deterministic ones in this context.

MIDACO is a new software based on such a purely stochastic approach for general MINLP problems. The intention of this paper is to evaluate the performance of MIDACO on a set of 100 non-convex MINLP problems and compare it to established MINLP software. Yet, to the best knowledge of the authors, this is the first time, that the performance of a purely stochastic approach on MINLP is directly compared to the performance of a set of deterministic approaches on a comprehensive set of benchmark problems.

The numerical results are surprising. Those reveal that MIDACO can obtain a significant higher percentage on global (or best known) optimal solutions than any of the tested deterministic MINLP software at a reasonable cpu-runtime (see Table 1 and Table 2). However, this study also shows, that MIDACO requires many more function evaluation than the deterministic solvers.

The high amount of function evaluation usually required by stochastic algorithms is often considered a knock-out argument for their application to cpu time expensive problems (like many real world applications). In case of MIDACO this argument is not fully effective. The MIDACO software features the option of massive parallelization of the problem function evaluation. Therefore this feature can be seen as a remedy, enabling the use of it even on very time consuming problems. The numerical results presented in this contribution also investigate the impact of massive parallelization on the performance of MIDACO on the same test set of 100 MINLP benchmarks. The results demonstrate, that MIDACO is even competitive to some of the deterministic SQP-based algorithms in terms of (serial processed) function evaluations, if parallelization is used.

This paper is structured as follows: Firstly, a general introduction to the software MIDACO is given. Secondly, extensive numerical results investigating the performance on 100 MINLP bench-

marks under several scenarios is presented, discussed and compared with established deterministic MINLP software. Next, a brief section is referencing some successful MIDACO utilizations on real world applications. Finally, some conclusions are drawn. Additionally to the numerical results section, two appendices are attached. Those provide detailed information on the individual benchmarks tested and the corresponding MIDACO performance.

# 2 The MIDACO Software

This section provides information on the software named MIDACO, which stands for *Mixed Integer Distributed Ant Colony Optimization*. It is a global optimization algorithm for black-box MINLP problems based on the ant colony optimization metaheuristic for continuous search domains proposed by Socha and Dorigo [18]. The intention of this section is to provide general information on the software and its usage, but not to give an introduction to ant colony optimization in general. Readers with interest in ant colony optimization (ACO) are requested to directly consult corresponding literature like Dorigo [6] and Blum [2] (for a general introduction) or Socha [17] (for the application of ACO on continuous and mixed integer search domains). Readers with a special interest in the underlying theoretical ACO algorithm of MIDACO can find a detailed description in [19] and [20]. Readers with a particular interest in the constraint handling technique of MIDACO will find comprehensive information in [21] and are kindly invited to contact the author directly.

MIDACO has been developed for a time period of over four years 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 depended random number generators. Uniform random numbers are generated by an internal implementation of a Xorshift generator [14] and transformed to normal distributed ones via the Box-Muller [1] method. MIDACO does not relax discrete optimization variables. A main focus of the software is its user friendliness and easy compilation, hence it is distributed within a single file for Fortran (`midaco.f`) and C (`midaco.c`) and in 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 MIDACO algorithm is a derivative free method and is therefore able to handle even discontinues problem functions. 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 features enables the algorithm to be executed even over a long time horizon without the necessity of any user interference. The software is intended for problems with up to some hundreds of variables and constraints. Interested readers can find more information on the MIDACO home page `www.midaco-solver.com`.

## 2.1 Reverse Communication and Distributed Computing

A key feature of MIDACO is its calling 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), but also enables the software to be enriched by 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 [16].

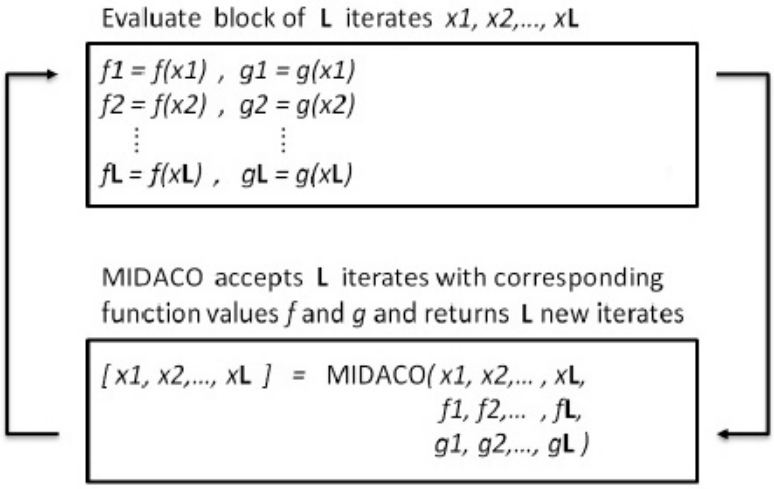


Figure 1: The reverse communication loop over a block of **L** iterates

Figure 1 illustrates the essential reverse communication loop over the function evaluation calls to $f(x)$ and $g(x)$ and the MIDACO code. Within one loop, a block of **L** iterates is evaluated regarding their objective function $f(x)$ and constraints $g(x)$. 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 evaluations. 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 3.4.1).

## 2.2 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 briefly described:

*Seed* - Initial seed for internal pseudo random number generator within MIDACO. The *Seed* determines the sequence of pseudo random numbers sampled by the generator. 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 esp. useful, if a run must be stopped out of some reason and should be restarted again.

*Qstart* - 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. 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 initial population of ants is specifically focused within these bounds.

*Autostop* - 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, that did not reveal an improvement in the objective function value. The greater *Autostop* is selected, the higher the chance of 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* - This parameter specifies a user given oracle parameter $\Omega$ [21] 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 [21]) starts to take place. This option can be useful for problems with difficult constraints where some background knowledge on the problem exists.

*Ants* - This parameter fixes the amount of iterates (also called ants) 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* - This parameter fixes the kernel size in a generation used by the Gauss distributions. The *Kernel* parameter must be used in combination with the *Ants* parameter and is intended to make MIDACO more efficiently on specific problems.

*Character* - 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 98 continuous and 2 integer variables. 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 (see http://www.midaco-solver.com/download.html). 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. 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).

# 3 Numerical Results

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 [15] and can freely be downloaded at

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

Please note, that many of these problems are originally taken form the GAMS library MINLPlib [9]. 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. Their difficulty widely ranges from *very easy* to *difficult*. 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. Further numerical results investigate the performance under consideration of its automatic stopping criteria. The impact of (massive) parallelization of the problem function calls is investigated in Section 3.4. An additional comparison of MIDACO to the MINLP solvers BONMIN [4] and COUENNE [5] is performed on a subset of 66 problems, which are provided in the GAMS library MINLPlib [9]. Finally a brief subsection illustrates the black-box capabilities of MIDACO, by demonstrating the performance on some test problems, which incorporate critical function properties.

All numerical results in this contribution refer to the Fortran77 version of MIDACO. The results in Section 3.2, Section 3.3 and Section 3.4 were computed by MIDACO version 3.0 on a computer with an Intel(R) Xeon(R) E5640 CPU with 2.67GHz clock rate. The results in Section 3.5 and Section 3.6 were computed by MIDACO version 2.0 on a computer with an Intel(R) Core(TM) i7 Q820 CPU with 1.73GHz clock rate. If not explicitly mentioned differently, MIDACO has been used by its default parameters (see Section 2.2).

## 3.1 Performance of SQP-based Algorithms

In a recent study by Exler et al. [8] eight sophisticated implementations of SQP-based algorithms for MINLP were presented and evaluated on this set of 100 benchmarks. Table 1 contains essential information (taken from Exler et al. [8], calculated on an Intel(R) Core(7) i7 processor with 3.16GHz) on the performance of these eight algorithms. The abbreviations of Table 1 are as follows:

| | | |
|---|---|---|
| $Algorithm$ | - | Name of the SQP-based algorithm for MINLP used in Exler et al. [8] |
| ***Optimal*** | - | Number of global (or best known) 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 global optimal solved problems |
| $Time_{mean}$ | - | Average number of function evaluations over global optimal solved problems |

Table 1: Performance of SQP-based algorithms on 100 MINLP benchmarks presented in [8]

| *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. [8]. Determining the success of an algorithm in finding a global (or best known) optimal solution is done in Exler et al. [8] by the following criteria:

$$\frac{|f(x,y) - f(x^*,y^*)|}{|f(x^*,y^*)|} < \epsilon, \tag{2}$$

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

$$\|g(x,y)\|_\infty < acc, \tag{3}$$

where $acc$ is some accuracy. In Exler et al. [8] a tolerance of $\epsilon = 0.01$ 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 [15]. In Exler et al. [8] 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.

## 3.2 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 (2) and (3)) like in Exler et al. [8] is applied. For the test runs of MIDACO a tolerance of $\epsilon = 0.01$ and an accuracy of $acc = 0.0001$ was used, which are identical to those tolerances as used in Exler et al. [8]. In the following, the term *global optimal* solution will always refer to the best known solution provided in Schittkowski [15].

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 (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. [8] stop by themselves and the optimality criteria is checked afterwards. The internal stopping criteria of MIDACO, named *Autostop*, is investigated separately in Section 3.3.

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.

Table 2 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 2 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 2: 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 2 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. Please note, that this implies that the MIDACO software is able to process millions of iterates within seconds on a standard computer. Regarding the random seed, the best run was obtained for $Seed = 5$.

Comparing the results of Table 2 with the ones obtained by the SQP-based algorithms presented in Table 1, 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.

### 3.3 The Automatic Stopping Criteria

The numerical results by MIDACO presented in Section 3.2 used exclusively external stopping criteria, in particular a maximal cpu time budget and the success in finding the global optimal solution. In general it is difficult for most stochastic algorithms to decide by themselves, when to stop an optimization run. MIDACO is no exception and the missing of an accurate stopping criteria is considered a small disadvantage of the algorithm. In practice, it is recommended for

users to execute MIDACO with a maximal available time budget. This procedure ensures the highest chance of global optimality.

Nevertheless, MIDACO offers the option of a heuristic stopping criteria based on the current progress of the algorithm. If the user activates the *Autostop* parameter (this is selecting it an integer greater than zero, see Section 2.2), MIDACO will stop by itself, if a number of *Autostop* internal restarts in sequence did not improve the best feasible solution. The algorithm will never stop (even if *Autostop* is activated), if no feasible solution at all has been found so far. This stopping criteria is rather primitive, but in contrast to a fixed number of iterations and alike, it takes into account the current progress of the algorithm.

A benefit of the *Autostop* parameter is that the user can scale this option to his or her specific needs by selecting either a small or a large value. A small value for *Autostop* has a higher probability to cause the algorithm to stop prematurely, but results in a faster runtime. A large value for *Autostop* will result in a higher probability in finding the global optimal solution, but will also increase the runtime.

In Table 3 six different MIDACO runs have been performed on the set of 100 MINLP benchmarks applying different *Autostop* values. Because the *Autostop* feature is only in effect, if a feasible solution is already found by MIDACO and the test set contains some highly constrained problems, the additional stopping criteria by a maximal evaluation budget is applied. This budget is chosen accordingly to the value of *Autostop*. As initial point, MIDACO assumes a random point for every individual test run. The abbreviations used in Table 3 are as follows:

| | | |
|---|---|---|
| $Autostop$ | - | Parameter value for automatic stopping criteria within MIDACO |
| $Maxeval$ | - | Maximum number of function evaluation for each problem |
| ***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 3: MIDACO performance regarding different *Autostop* values and evaluation budgets

| $Autostop$ | $Maxeval$ | ***Optimal*** | $Feasible$ | $Eval_{mean}$ | $Time_{mean}$ |
|---|---|---|---|---|---|
| 1 | 100000 | **60** | 92 | 24,457 | 0.04 |
| 5 | 500000 | **77** | 95 | 122,254 | 0.43 |
| 10 | 1000000 | **82** | 98 | 256,931 | 0.90 |
| 50 | 5000000 | **87** | 98 | 1,232,462 | 4.71 |
| 100 | 10000000 | **91** | 98 | 2,656,601 | 10.05 |
| 500 | 50000000 | **96** | 98 | 13,776,828 | 50.79 |

Table 3 reflects the effect on the MIDACO performance regarding the different values for the *Autostop* parameter. For $Autostop = 1$ a low rate of 60 global optimal solutions is found with an average of 24,457 evaluation and an average time of just 0.04 seconds. For $Autostop = 500$ a high rate of 96 global optimal solutions is obtained with an average of around 13,7 million evaluation and an average time of 50.79 seconds. The compromise between a fast runtime and a high rate of global optimality by using different values for *Autostop* can be well observed in Table 3.

## 3.4 The Impact of Parallelization

As seen in Section 3.2 the MIDACO software is able to process millions of iterates within seconds. 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 2.1) 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 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 3.2, the success criteria (2) is applied to stop MIDACO, in case it reveals the global optimal solution before performing the maximal budget of blocks. Table 4 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 4 are as follows:

| | | |
|---|---|---|
| **L** | - | Parallelization factor for MIDACO (see Section 2.1) |
| ***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 4: 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 4 demonstrate the significant impact of the parallelization factor **L** on the MIDACO performance. Like expected, the results corresponding to a parallelization factor **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 **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 1) in terms of function evaluation, if (massive) parallelization capabilities are available.

### 3.4.1 A Note on Computational Expensive Applications

As mentioned in the introduction, the high amount of function evaluation usually required by stochastic algorithms is often used as knock-out argument regarding their application on computational expensive applications. In Section 3.4 the remedy of parallelization of the time costly function evaluation was presented, which enables the use of MIDACO even on such costly applications.

Here a small reflection on the nature of cpu time expensive applications and its consequences should be undertaken. It is reasonable to assume, that if an evaluation of a function (depending on not more than some hundred variables) is requiring a lot of computational effort, it is a complex function. In esp. this means, the longer the computational evaluation time, the higher the complexity. Very complex functions are more likely to include critical properties such as high non-convexity, discontinuities, flat spots or even stochastic distortions.

Most deterministic approaches have great difficulties in dealing with such properties. Stochastic approaches like MIDACO in contrary are mostly black-box optimizers and therefore capable to deal well with such characteristics. As a consequence, the application of a stochastic approach exploiting (massive) parallelization seems to be a more reasonable and promising choice on computational expensive applications in general. However, in case the specific nature of a computational expensive application is well known and a proper algorithm is available, this argument is not valid.

## 3.5 Comparison with BONMIN and COUENNE on GAMS benchmarks

In addition to the comparison of MIDACO to the set of SQP based MINLP solvers given in Section 3.1, a further comparison to the established MINLP solvers BONMIN [4] and COUENNE [5] 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 [10]), 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 [11] 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 9) is considered for evaluating purposes. This subset represent those problems from the 100 MINLP benchmarks, that are originally taken from the GAMS MINLPlib [9] 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 3.3) was activated. A value of 50 was chosen for the *Autostop* parameter (see Table 3), which seems to provide a good balance between solution quality and cpu-runtime, based on the results reported in Table 3. 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 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 can be found in the Appendix B, Table 13. Function evaluation are not reported in Table 5, as those information is not consistently reported within the GAMS environment for BONMIN and COUENNE. In case of MIDACO, Table 2 and Table 3 do already

give 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: Comparison of 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 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, 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 assumingly due to numerical difficulties and software bugs. However, this observation seems to put the numerical relevance of a theoretical convergence proof, which MIDACO lacks of, into further perspective here.

### 3.6 Test Problems with Critical Function Properties

In order to demonstrate the black-box capabilities of MIDACO, five mixed integer test problems, incorporating critical function properties, are considered here. These critical properties are namely: stochastic noise, flat spots, non-convexity and non-differentiability. With exception to non-convexity, these function properties are known to be very undesirable for most deterministic optimization algorithms and are likely to be a knock-out argument for their usage. In contrast to this, MIDACO (based on a stochastic metaheuristic) can deal well with such critical function properties.

Table 6 lists the five considered test problems regarding their name, objective function and main characteristic property. For all problems, the dimension of continuous variables $x$ and discrete variables $y$ is considered to be 15. The variables $x$ are considered to range within $[-10, 10]$ and analog the variables $y$ are considered to range within the discrete set $\{-10, 10\}$. All problems have their global optimal solution at zero. The first problem (CRIT1) does add stochastic noise in form of uniformly distributed random numbers ($\tilde{u}, \hat{u} \in [0, 1]^{15}$) to the objective function. The second problem (CRIT2) applies a Gauss bracket on the continuous variables $x$, which rounds them to the nearest integer and therefore implies flat spots in the continuous search domain. The third problem (CRIT3) is the well known and highly non-convex rastringin function [23] applied here on the continuous as well as the discrete variables. The fourth problem (CRIT4) considers a representation of the Weierstrass function [25], which is at no point differentiable. The fifth problem (CRIT5) is simply the sum over problem one to four and therefore contains (in parts) all the above critical properties.

Table 6: Five test problems with critical function properties

| *Name* | *Objective Function* | *Property* |
|---|---|---|
| CRIT1 | $f_1(x,y) = \sum_{i=1}^{15} x_i^2 + \lvert\tilde{u}_i \cdot x_i\rvert + y_i^2 + \lvert\hat{u}_i \cdot y_i\rvert$ | stochastic noise ($\tilde{u}_i, \hat{u}_i \in [0,1]$) |
| CRIT2 | $f_2(x,y) = \sum_{i=1}^{15} [x_i]^2 + y_i^2$ | flat spots |
| CRIT3 | $f_3(x,y) = 150 \sum_{i=1}^{15} x_i^2 - 10cos(2\pi x_i) + y_i^2 - 10cos(2\pi y_i)$ | non-convexity |
| CRIT4 | $f_4(x,y) = \sum_{i=1}^{15} (x_i \sum_{k=1}^{10} \frac{2^k sin(2^k x_i)}{3^k})^2$ | non-differentiability |
| CRIT5 | $f_5(x,y) = f_1(x,y) + f_2(x,y) + f_3(x,y) + f_4(x,y)$ | |

MIDACO has been tested on every problem with 10 test runs, using a different random seed and the lower bounds as starting point. Every test run was stopped, if the success criteria of reaching the global optimal objective function value with a precision of $10^{-4}$ was reached. Table 7 reports the average number of function evaluation and average time on every problem. MIDACO did reach the global solution in all cases.

Table 7: Results on critical test problems

| *Name* | $Eval_{aver}$ | $Time_{aver}$ |
|---|---|---|
| CRIT1 | 3241433 | 15.24 |
| CRIT2 | 4608 | 0.02 |
| CRIT3 | 909501 | 5.26 |
| CRIT4 | 694113 | 6.34 |
| CRIT5 | 1436361 | 15.16 |

From the results in Table 7 it can be concluded, that MIDACO is well capable of handling problems with critical function properties. All problems could be solved in a reasonable time for a moderate problem dimension of 30 variables in total. The stochastic noise in problem CRIT4 seems to be the biggest challenge on this test set for MIDACO, while the presence of flat spots in problem CRIT2 seem to have only a very low impact on the MIDACO performance.

# 4 Real World Applications

While the focus of this contribution is on the MIDACO performance on a set of 100 MINLP benchmark problems (see Section 3), this section will provide further references to successful application of the MIDACO algorithm to real world problems done in the past. The sole purpose of this section is to confirm, that MIDACO is not only promising on academic benchmark problems, but also on challenging real world applications.

Table 8 provides information on several applications where beta versions of MIDACO (please note, that the very first beta version was named ACOmi in [19] and [20]) where used. Table 8 lists brief information on the name and area of the application, its dimensions and the solution quality obtained by MIDACO. For every problem a literature (or web-) reference to the published MIDACO solution is given. Further information on the applications and the solutions can also be found in those references. Table 8 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 in total of the application |
| $n_{int}$ | - | Number of integer decision variables of the application |
| $m$ | - | Number of constraints in total of the application |
| $m_{eq}$ | - | Number of equality constraints of the application |
| *Reference* | - | Literature or (web-) reference for the application |

Table 8: Real world applications solved by MIDACO (with published solutions)

| *Area* | *Name* | ***Solution*** | $n$ | $n_{int}$ | $m$ | $m_{eq}$ | *Reference* |
|---|---|---|---|---|---|---|---|
| Chem. Eng. | TEP | **best** | 43 | 7 | 11 | 1 | [20] |
| Chem. Eng. | WWT.COST.1 | **best** | 4 | 0 | 0 | 0 | [20] |
| Chem. Eng. | WWT.COST.2 | **best** | 8 | 0 | 0 | 0 | [20] |
| Chem. Eng. | WWT.COST.3 | **best** | 13 | 1 | 0 | 0 | [20] |
| Chem. Eng. | TP4 | ***new* best** | 52 | 2 | 38 | 35 | [21] |
| Chem. Eng. | TP5 | ***new* best** | 113 | 3 | 71 | 67 | [21] |
| Space | GTOC1 | ***new* best** | 8 | 0 | 6 | 0 | [7] |
| Space | Cassini2 | ***new* best** | 22 | 0 | 0 | 0 | [7] |
| Space | Satellite | **first solution** | 5 | 2 | 1 | 0 | [22] |
| Aero Space | Heat Shield | ***new* best** | 31 | 11 | 2 | 0 | [19] |
| Aero Space | F8-Aircraft | **best** | 6 | 0 | 183 | 3 | [21] |
| Robotics | Camera | **first solution** | 45 | 0 | 6 | 6 | [13] |

The solution qualities referred to in Table 8 illustrate the potential of MIDACO. In most cases the best known or even a new best known solution could be achieved. The application in Table 8 do also include purely continuous optimization (NLP) problems. As the MIDACO algorithm is constructed for general MINLP problems, it can also be applied on this kind of optimization problems. The results on the space applications (GTOC1 and Cassini2), which are known to be difficult, indicate, that MIDACO can be a promising choice for NLP problems as well.

# 5 Conclusions

For the first time, a purely stochastic algorithm for MINLP (named MIDACO) was numerically evaluated on a comprehensive set of 100 benchmark problems. The extensive numerical results revealed the strength of MIDACO in terms of robustness, global optimality and reasonable cpu-runtime performance (see Table 2). The comparison with results by several established MINLP software (see Table 1 and Table 5) showed, that MIDACO is able to significantly outperform those regarding the number of global optimal solutions obtained. Even though MIDACO requires many function evaluation (due to its stochastic nature), it can still be faster than other MINLP software, if problem function evaluation are computational inexpensive (see Table 1, Table 2 and Table 5). The platform independent (massive) parallelization feature of MIDACO enables the software to be even promising on cpu time expensive applications (see Table 4). Furthermore the black-box capabilities of MIDACO due offer an intriguing advantage over deterministic MINLP solvers, that require gradients or smooth function properties. In conclusion, MIDACO is considered an innovative and powerful new software for MINLP, available in several major programming languages.

## Acknowledgments

The authors would like to thank K. Schittkowski, O. Exler and T. Lehmann for generously providing the test set of MINLP benchmarks and many supportive help. Special thanks go to K. Schittkowski for inspiring important implementation features like the reverse communication and the parallelization option for MIDACO. The development has been supported by the project *"Nonlinear mixed-integer-based Optimisation Technique for Space Applications" (ESTEC/Contract No. 21943/08/NL/ST)* co-funded by ESA Networking Partnership Initiative, Astrium Limited (Stevenage, UK) and the School of Mathematics, University of Birmingham, UK.

## 6 Appendix A

Individual results by MIDACO for a test run on the set of 100 non-convex MINLP benchmarks provided by Schittkowski [15] are reported. Table 9 lists the individual benchmark names as reported in [15] and addresses a library number to them.

Table 9: 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 2 (where 10 different runs on the full library where performed) the random *Seed* 2.2 was fixed to 9, 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 3.2). Besides the cpu time budget, the success criteria (2) in finding a global optimal solution was applied with $\epsilon = acc = 0.0001$. Table 10 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 $L_\infty$-norm over all violations |

Table 10: 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 | 0.0000000000 |
| | 2 | 6 | 0.0 | 4 | 4 | 4 | 0 | -20.0000 | -20.0000 | 0.0000000000 |
| | 3 | 20727 | 0.0 | 5 | 3 | 7 | 0 | 3.5000 | 3.5026 | 0.0000000000 |
| | 4 | 13798 | 0.0 | 4 | 3 | 3 | 0 | -40.9566 | -40.6317 | 0.0000000000 |
| | 5 | 59 | 0.0 | 4 | 4 | 3 | 0 | -38.0000 | -38.0000 | 0.0000000000 |
| | 6 | 4242 | 0.0 | 7 | 4 | 4 | 0 | 694.9027 | 696.3917 | 0.0000000000 |
| | 7 | 89 | 0.0 | 7 | 7 | 4 | 0 | 700.0000 | 700.0000 | 0.0000000000 |
| | 8 | 24443 | 0.0 | 10 | 6 | 8 | 0 | 37.2195 | 37.5528 | 0.0000000000 |
| | 9 | 717 | 0.0 | 10 | 10 | 8 | 0 | 43.0000 | 43.0000 | 0.0000000000 |
| | 10 | 80 | 0.0 | 25 | 13 | 10 | 0 | -304723942.9203 | -301895650.3390 | 0.0000000000 |
| | 11 | 31677 | 0.1 | 7 | 3 | 2 | 0 | 1.0000 | 1.0059 | 0.0000000000 |
| | 12 | 27922 | 0.0 | 7 | 3 | 4 | 0 | -2.7183 | -2.6941 | 0.0000000000 |
| | 13 | 96735 | 0.1 | 7 | 3 | 4 | 0 | -8980002.7000 | -8948767.1685 | 0.0000000000 |
| | 14 | 1 | 0.0 | 2 | 2 | 2 | 0 | -56.9375 | -56.9375 | 0.0000000000 |
| | 15 | 228 | 0.0 | 5 | 5 | 3 | 0 | 0.1004 | 0.1004 | 0.0000000000 |
| | 16 | 6247 | 0.0 | 5 | 2 | 6 | 0 | -30665.5387 | -30372.7513 | 0.0000000000 |
| | 17 | 13 | 0.0 | 2 | 1 | 2 | 0 | -2.4444 | -2.4368 | 0.0000000000 |
| | 18 | 472043 | 0.7 | 3 | 2 | 3 | 1 | 12.4697 | 12.4697 | 0.0000863230 |
| | 19 | 465482 | 0.6 | 8 | 5 | 3 | 3 | 5.9642 | 6.0164 | 0.0000615147 |
| | 20 | 35 | 0.0 | 2 | 2 | 2 | 0 | 16.0000 | 16.0000 | 0.0000000000 |
| | 21 | 58 | 0.0 | 2 | 2 | 0 | 0 | 0.7200 | 0.7200 | 0.0000000000 |
| - | 22 | 245937264 | 300.0 | 8 | 2 | 9 | 4 | 5.4709 | 29.4678 | 0.0000884825 |
| | 23 | 43 | 0.0 | 2 | 2 | 0 | 0 | 1.7703 | 1.7703 | 0.0000000000 |
| | 24 | 311 | 0.0 | 3 | 3 | 2 | 0 | 4.0000 | 4.0000 | 0.0000000000 |
| | 25 | 8188 | 0.0 | 3 | 2 | 3 | 0 | 23.4497 | 23.6467 | 0.0000000000 |
| | 26 | 148 | 0.0 | 10 | 10 | 0 | 0 | -43.1343 | -43.1343 | 0.0000000000 |
| | 27 | 41 | 0.0 | 2 | 2 | 2 | 0 | -310.8000 | -310.8000 | 0.0000000000 |
| | 28 | 113 | 0.0 | 3 | 3 | 3 | 0 | -431.0000 | -427.8000 | 0.0000000000 |
| | 29 | 119 | 0.0 | 4 | 4 | 4 | 0 | -481.2000 | -481.2000 | 0.0000000000 |
| | 30 | 159 | 0.0 | 5 | 5 | 5 | 0 | -585.2000 | -582.8000 | 0.0000000000 |
| | 31 | 41158 | 0.1 | 8 | 5 | 3 | 3 | -40358.1500 | -40256.3555 | 0.0000456870 |
| | 32 | 157 | 0.0 | 3 | 3 | 1 | 0 | 1.0000 | 1.0000 | 0.0000000000 |
| | 33 | 17 | 0.0 | 2 | 2 | 0 | 0 | 0.7031 | 0.7031 | 0.0000000000 |
| | 34 | 278 | 0.0 | 7 | 7 | 7 | 0 | -1100.4000 | -1093.0000 | 0.0000000000 |
| | 35 | 165 | 0.0 | 6 | 6 | 6 | 0 | -778.4000 | -772.4000 | 0.0000000000 |
| | 36 | 259 | 0.0 | 8 | 8 | 8 | 0 | -1098.4000 | -1088.6000 | 0.0000000000 |
| | 37 | 197257 | 0.4 | 16 | 5 | 8 | 0 | 230.9222 | 233.1332 | 0.0000000000 |
| | 38 | 5690 | 0.0 | 3 | 2 | 2 | 0 | -5.6848 | -5.6746 | 0.0000000000 |
| | 39 | 533239 | 0.7 | 8 | 4 | 9 | 4 | 6.0582 | 6.0582 | 0.0000880749 |
| | 40 | 620 | 0.0 | 9 | 9 | 9 | 0 | -1125.2000 | -1114.4000 | 0.0000000000 |
| | 41 | 405 | 0.0 | 10 | 10 | 10 | 0 | -1033.2000 | -1023.0000 | 0.0000000000 |
| | 42 | 70829 | 0.1 | 8 | 4 | 4 | 4 | 1.0000 | 1.0024 | 0.0000218749 |
| | 43 | 11231785 | 18.5 | 14 | 3 | 13 | 13 | 0.2545 | 0.2543 | 0.0000833918 |
| | 44 | 79347 | 0.1 | 6 | 3 | 6 | 0 | 6.0097 | 6.0536 | 0.0000000001 |
| | 45 | 40632 | 0.1 | 11 | 5 | 14 | 1 | 73.0357 | 73.7594 | 0.0000800100 |
| | 46 | 149590 | 0.3 | 17 | 8 | 23 | 2 | 68.0100 | 68.6351 | 0.0000834528 |
| | 47 | 185537 | 0.2 | 5 | 3 | 5 | 2 | 7.6672 | 7.6672 | 0.0000888507 |
| | 48 | 30249 | 0.0 | 3 | 1 | 3 | 0 | 1.0765 | 1.0820 | 0.0000959807 |
| | 49 | 197477 | 0.2 | 7 | 4 | 9 | 0 | 4.5796 | 4.6250 | 0.0000000000 |
| | 50 | 13470477 | 18.7 | 11 | 8 | 7 | 3 | -0.9435 | -0.9435 | 0.0000843793 |

Table 11: 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* |
|---|---|---|---|---|---|---|---|---|---|---|
| | 51 | 150 | 0.0 | 2 | 2 | 4 | 0 | 31.0000 | 31.0000 | 0.0000000000 |
| | 52 | 1222 | 0.0 | 2 | 1 | 3 | 0 | -17.0000 | -16.8306 | 0.0000000000 |
| | 53 | 126780 | 0.2 | 9 | 3 | 7 | 3 | -1.9231 | -1.9191 | 0.0000401917 |
| | 54 | 1204227 | 2.3 | 18 | 12 | 8 | 5 | 0.8462 | 0.8436 | 0.0000995025 |
| | 55 | 3 | 0.0 | 4 | 4 | 0 | 0 | 1.0000 | 1.0021 | 0.0000000000 |
| | 56 | 13930 | 0.0 | 4 | 2 | 2 | 0 | 1.3634 | 1.3726 | 0.0000000000 |
| | 57 | 11501 | 0.0 | 6 | 4 | 1 | 1 | 1.0000 | 1.0003 | 0.0000775729 |
| | 58 | 70829 | 0.1 | 8 | 4 | 4 | 4 | 1.0000 | 1.0024 | 0.0000218749 |
| - | 59 | 120829756 | 300.0 | 24 | 9 | 34 | 13 | 128918.0000 | 134407.2581 | 0.0000996744 |
| | 60 | 6119043 | 18.5 | 24 | 24 | 35 | 0 | 19.6000 | 19.6000 | 0.0000000000 |
| | 61 | 6431077 | 19.3 | 24 | 24 | 35 | 0 | 8.6000 | 8.6000 | 0.0000000000 |
| | 62 | 40895454 | 146.9 | 35 | 35 | 44 | 0 | 10.3000 | 10.3000 | 0.0000000000 |
| | 63 | 5975068 | 23.9 | 48 | 48 | 53 | 0 | 16.3000 | 16.3000 | 0.0000000000 |
| | 64 | 184918 | 1.1 | 20 | 13 | 9 | 0 | 8.4806 | 8.5625 | 0.0000000000 |
| | 65 | 206575 | 1.7 | 20 | 13 | 9 | 0 | 3.5392 | 3.5501 | 0.0000000000 |
| x | 66 | 87406151 | 300.0 | 35 | 19 | 18 | 17 | -1.4304 | -1.4365 | 0.0203979103 |
| | 67 | 69575 | 0.1 | 2 | 1 | 2 | 1 | -246.0000 | -243.8568 | 0.0000393151 |
| | 68 | 3718 | 0.0 | 4 | 2 | 3 | 0 | 7197.7271 | 7203.1783 | 0.0000053846 |
| | 69 | 15886 | 0.0 | 4 | 3 | 5 | 1 | 28.2426 | 28.4142 | 0.0000312663 |
| | 70 | 28 | 0.0 | 5 | 5 | 1 | 0 | 281.0000 | 281.0000 | 0.0000000000 |
| | 71 | 4628 | 0.0 | 4 | 4 | 3 | 0 | 2.0000 | 2.0000 | 0.0000000000 |
| | 72 | 21236 | 0.0 | 2 | 2 | 1 | 0 | -6.0000 | -6.0000 | 0.0000000000 |
| | 73 | 50668 | 0.1 | 6 | 3 | 4 | 0 | -4574.0000 | -4532.5531 | 0.0000000000 |
| | 74 | 205239 | 0.3 | 7 | 2 | 13 | 0 | -333.8900 | -333.7328 | 0.0000901816 |
| | 75 | 111 | 0.0 | 5 | 5 | 1 | 0 | -4500.0000 | -4497.5000 | 0.0000000000 |
| | 76 | 267 | 0.0 | 6 | 6 | 2 | 0 | -9.2500 | -9.2500 | 0.0000000000 |
| | 77 | 116161 | 0.3 | 13 | 13 | 10 | 0 | -7.0000 | -7.0000 | 0.0000000000 |
| | 78 | 70826 | 0.1 | 6 | 6 | 5 | 0 | -7.0000 | -7.0000 | 0.0000000000 |
| | 79 | 471 | 0.0 | 10 | 10 | 11 | 0 | -110.0000 | -110.0000 | 0.0000000000 |
| | 80 | 96 | 0.0 | 10 | 10 | 5 | 0 | 471.0000 | 471.0000 | 0.0000000000 |
| | 81 | 343156 | 1.0 | 24 | 24 | 20 | 0 | -29605.0000 | -29311.0000 | 0.0000000000 |
| | 82 | 3139 | 0.0 | 10 | 10 | 5 | 0 | -12.8116 | -12.6946 | 0.0000000000 |
| | 83 | 123064 | 0.3 | 20 | 20 | 20 | 0 | -20.5900 | -20.3932 | 0.0000000000 |
| | 84 | 144 | 0.0 | 3 | 3 | 10 | 0 | -80.5000 | -80.5000 | 0.0000000000 |
| | 85 | 78 | 0.0 | 8 | 8 | 12 | 0 | 2.3000 | 2.3000 | 0.0000000000 |
| | 86 | 3236690 | 9.3 | 24 | 24 | 24 | 0 | 8.3000 | 8.3000 | 0.0000000000 |
| | 87 | 15715534 | 53.8 | 35 | 35 | 30 | 0 | 10.3000 | 10.4000 | 0.0000000000 |
| | 88 | 26157808 | 97.3 | 48 | 48 | 36 | 0 | 14.6000 | 14.7000 | 0.0000000000 |
| | 89 | 7268 | 0.0 | 6 | 6 | 8 | 0 | 112235.0000 | 112235.0000 | 0.0000000000 |
| | 90 | 4585740 | 18.4 | 48 | 48 | 53 | 0 | 16.3000 | 16.3000 | 0.0000000000 |
| | 91 | 398513 | 1.6 | 48 | 48 | 54 | 0 | 48.0667 | 48.0667 | 0.0000000000 |
| | 92 | 306034 | 0.4 | 8 | 4 | 7 | 2 | 2.9250 | 2.9262 | 0.0000379279 |
| | 93 | 3132273 | 10.9 | 35 | 14 | 44 | 8 | 14.1897 | 14.3309 | 0.0000990643 |
| | 94 | 232 | 0.0 | 16 | 16 | 7 | 0 | 13.0000 | 13.0000 | 0.0000000000 |
| | 95 | 24412 | 0.0 | 5 | 3 | 7 | 0 | 3.5000 | 3.5177 | 0.0000000000 |
| | 96 | 94 | 0.0 | 5 | 5 | 1 | 0 | 8.0000 | 8.0000 | 0.0000000000 |
| | 97 | 38993 | 0.1 | 10 | 10 | 1 | 0 | 300.0000 | 302.0000 | 0.0000000000 |
| | 98 | 81355 | 0.4 | 20 | 20 | 3 | 0 | 0.1318 | 0.1329 | 0.0000000000 |
| | 99 | 129314 | 1.4 | 50 | 50 | 3 | 0 | 0.4052 | 0.4090 | 0.0000000000 |
| | 100 | 599873 | 12.1 | 100 | 100 | 3 | 0 | 1.0973 | 1.0975 | 0.0000000000 |

Table 12: Summary of results presented in Table 10

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

# 7 Appendix B

Individual results by the MINLP solvers BONMIN [4], COUENNE [5] and MIDACO for a set of 66 MINLP problems form the GAMS MINLPlib [9] 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 3.3) 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 10. A summary of the individual results presented in Table 13 can be found in Section 3.5 Table 5.

Table 13: Individual results by BONMIN, COUENNE and MIDACO on 66 MINLP problems

| *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 |
| 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 |

Table 14: Continuation of Table 13

| *Problem* | *Solver* | *Optimal* | *Feasible* | *Time* |
|---|---|---|---|---|
| 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 |
| 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 |

Table 15: Continuation of Table 13

| *Problem* | *Solver* | *Optimal* | *Feasible* | *Time* |
|---|---|---|---|---|
| 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 |
| 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 |
| 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 |

# References


[1] G.E.P. Box and M.E. Muller, *A Note on the Generation of Random Normal Deviates*, Ann. Math. Statist. 29(2) (1958), pp. 610–611

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

[3] M.R. Bussieck and S. Vigerske, *MINLP Solver Software*, J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh and J.C. Smith, Wiley Encyclopedia of Operations Research and Management Science, John Wiley and Sons, Inc., New York, 2010

[4] COIN-OR (Project Manager P. Bonami), *Basic Open-source Nonlinear Mixed INteger programming*; software available at http://www.coin-or.org/Bonmin/

[5] COIN-OR (Project Manager P. Belotti), *Convex Over and Under ENvelopes for Nonlinear Estimation*; software available at http://www.coin-or.org/Couenne/

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

[7] 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

[8] O. Exler, T. Lehmann and K. Schittkowski, *A Comparative Study of SQP-Type Algorithms for Nonlinear and Nonconvex Mixed-Integer Optimization*, preprint (2010), submitted for publication. Available at http://www.ai7.uni-bayreuth.de/minlp_comp_study.htm

[9] *GAMS MINLPlib - A collection of Mixed Integer Nonlinear Programming models*. Washington, DC, USA; software available at http://www.gamsworld.org/minlp/minlplib.htm

[10] *Matt Saltzman: Informs 2010 Annual Meeting Blogposts*. Austin, Texas, USA. Available at http://meetings2.informs.org/Austin2010/blog/?p=88

[11] *GAMS (General Algebraic Modeling System)*. Washington, DC, USA; software available at http://www.gams.com/

[12] I.E. Grossmann, *Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques*, Optim. Eng. 3(3) (2002), pp. 227–252

[13] M. Hänel, S. Kuhn, D. Henrich, L. Grüne, and J. Pannek, *Optimal Camera Placement to measure Distances Conservativly Regarding Static and Dynamic Obstacles*, preprint (2011). Available at http://arxiv.org/abs/1105.3270

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

[15] 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

[16] 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

[17] K. Socha, *ACO for Continuous and Mixed-Variable Optimization*, Lect. Notes Comput. Sc. Vol. 3172, Springer, Berlin, 2004, pp. 25–36

[18] K. Socha and M. Dorigo, *Ant colony optimization for continuous domains*, Eur. J. Oper. Res. 85(2008), pp. 1155–1173

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

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

[21] M. Schlüter and M. Gerdts, *The Oracle Penalty Method*, J. Global Optim. 47(2) (2010), pp. 293–325

[22] A.T. Takano and B.G. Marchand, *Optimal Constellation Design for Space Based Situational Awareness Applications*, Astrodynamics Specialists Conference (AAS/AIAA), Paper No. AAS11-543, Girdwood, AK, 2011. Available at http://www.ae.utexas.edu/~marchand/AAS11-543.pdf

[23] A. Törn and A. Zilinskas, *Global Optimization*, Lect. Notes Comput. Sc. Vol. 350, Springer, 1989

[24] 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. Comput. 19(3) (2007), pp. 328–340

[25] K. Weierstrass, *Abhandlungen aus der Functionenlehre*, Julius Springer, Berlin (1886)