Schlueter M.★
Munetomo M.★

June 2016

http://www.midaco-solver.com

— User Manual Version 5.0 —

(See the release notes for changes regarding Version 6.0)

Abstract

MIDACO is a **general purpose software** to solve numerical optimization problems. This document provides a comprehensive introduction to MIDACO. Along with a brief description of the algorithmic background of MIDACO, focus will be given on the problem formulation, stopping criteria and MIDACO's parameter handling. Information on multi-objective optimization and parallelization with MIDACO are given in dedicated sections. For users with advanced challenges, a separate section with "tips and tricks" is presented. A list of all IFLAG messages used by MIDACO is included at the end of the document.

Quick Jump Menu

- Introduction
- Optimization Problem
- MIDACO Screen and Solution
- MIDACO Stopping Criteria
- MIDACO Parameter
- Multi-Objective Optimization
- Parallelization
- Tips & Tricks
- IFLAG Messages

★ IIC, Hokkaido University, Japan. {schlueter@midaco-solver.com, munetomo@iic.hokudai.ac.jp}

Contents

Overview **3**

Introduction **4**

1 Optimization Problem **8**
1.1 Problem Dimensions, Bounds and Starting Point 9
1.2 Problem Function Call 10
1.3 Passing Additional Input/Output Arguments 11
1.4 Verifying a Problem Implementation 11

2 MIDACO Screen and Solution **12**
2.1 PRINTEVAL and SAVE2FILE 14
2.2 Solution History File 14

3 MIDACO Stopping Criteria **15**
3.1 Hard Limit Criteria 15
3.1.1 MAXTIME 15
3.1.2 MAXEVAL 15
3.2 Algorithmic Criteria 16
3.2.1 FSTOP 16
3.2.2 ALGOSTOP 16
3.2.3 EVALSTOP 16
3.3 Example Scenarios 17
3.3.1 Single Evaluation 17
3.3.2 CPU-Time expensive application 17
3.3.3 CPU-Time cheap application 18
3.3.4 Infinite Run 18

4 MIDACO Parameter **19**
4.1 PARAM(1) : ACCURACY 19
4.2 PARAM(2) : SEED 19
4.3 PARAM(3) : FSTOP 19
4.4 PARAM(4) : ALGOSTOP 19
4.5 PARAM(5) : EVALSTOP 20
4.6 PARAM(6) : FOCUS 20
4.7 PARAM(7) : ANTS 20
4.8 PARAM(8) : KERNEL 21
4.9 PARAM(9) : ORACLE 21
4.10 PARAM(10) : PARETOMAX 22
4.11 PARAM(11) : EPSILON 22
4.12 PARAM(12) : CHARACTER 23

5 Multi-Objective Optimization **24**
5.1 Constructing an overall target function . . . 25
5.1.1 Single Target . . . 25
5.1.2 Weighted Sum . . . 26
5.1.3 Weighted Distance Function . . . 26
5.1.4 Additional Constraints . . . 28
5.1.5 Auto-Scaling . . . 29
5.2 PlotTool . . . 29

6 Parallelization **30**
6.1 Running MIDACO in parallel . . . 31
6.2 Parallelization overhead . . . 31

7 Tips & Tricks **32**
7.1 Constraint Handling . . . 32
7.2 Highly nonlinear problems . . . 33
7.3 Large-Scale Problems . . . 33
7.4 CPU-Time expensive applications . . . 33
7.5 Solving non-linear equation systems . . . 34
7.6 Joining multiple pareto front files . . . 34
7.7 Plotting the history file . . . 35
7.8 Plotting variable relationships . . . 35
7.9 Multi-modal optimization . . . 36
7.10 Submitting several starting points . . . 36
7.11 Parallel-Overclocking with MIDACO . . . 37

8 IFLAG Messages **38**
8.1 Solution Messages (IFLAG = $1 \sim 9$) . . . 38
8.2 Warning Messages (IFLAG = $10 \sim 99$) . . . 38
8.3 Error Messages (IFLAG = $100 \sim 999$) . . . 39

Acknowledgement **40**

References **40**

Overview

The Key facts on MIDACO:

- **MIDACO is a solver for general optimization problems**
  - Single- and multi-objective optimization
  - Continuous, integer and mixed integer variables
  - Constrained and unconstrained problems
- **Large scale capability**
  - MIDACO solves problems with up to **100,000** variables
  - MIDACO can handle up to thousands of constraints
- **Parallelization**
  - MIDACO features various parallelization schemes in several languages
  - Capable of massive parallelization with thousands of cores/threads
  - Intended for CPU-time intensive applications (e.g. process simulations)
- **Languages**
  - Excel, Java, VB, C#, R, Matlab, Octave, Python, C/C++, Fortran, ...
- **Source code**
  - Compact ANSI-C "midaco.c" and Fortran "midaco.f" source code
  - Completely self-sufficient source code (no third-party dependencies)
- **Usage and embedding**
  - Very easy to use and integrate (embed) into external software/algorithms
- **Applications**
  - (Aero)space, chemical engineering, finance, bio-technology, telecommunication, ...
- **Record solutions**
  - MIDACO holds several records on interplanetary space trajectory benchmarks
- **Background**
  - Developed in collaboration with the European Space Agency (ESA)
  - Extended with support of the Japanese Space Exploration Agency (JAXA)

Introduction

MIDACO-SOLVER is a software tool for numerical optimization. The MIDACO algorithm is constructed as **general-purpose solver** for single- as well as multi-objective optimization problems. A special feature of MIDACO is its capability to handle (constrained) mixed integer nonlinear programming (MINLP) problems. The term *mixed integer* refers here to optimization problems where some decision variables are of continuous type (like 1.23 or 4.56) while others are of discrete type (like 1, 2 or 3). The mathematical formulation of the general multi-objective MINLP considered by MIDACO is stated as follows:

$$\begin{aligned} &\text{Minimize} \quad f_1(x),\ f_2(x),...,\ f_O(x) \\ \\ &\text{subject to:} \quad g_i(x) = 0, \quad i = 1,...,m_e \\ &\qquad\qquad\ \ g_i(x) \geq 0, \quad i = m_e+1,...,m \\ &\qquad\qquad\ \ x_l \leq x \leq x_u \ \ \text{(box constraints)} \end{aligned} \tag{1}$$

In (1), the vector $f_{1,...,O}(x)$ denotes the objective functions and the vector $g_{1,...,m}(x)$ denotes the constraint functions. Without loss of generality, all objectives are subject to minimization. The first $1,...,m_e$ values of the constraint vector $g(x)$ represent equality constraints, while the remaining $m_e+1,...,m$ values represent in-equality constraints. The vector x of decision variables contains continuous variables as well as discrete variables (also called *integer*, *categorical* or *combinatorial* variables), whereas the continuous ones are stored first and the discrete ones are stored last. Furthermore, some box constraints as lower bounds x_l and upper bounds x_u are assumed for the decision variables x.

MIDACO solves above multi-objective MINLP (1) by combining an extended evolutionary *Ant Colony Optimization* (ACO) [21] algorithm with the *Oracle Penalty Method* [23] for constrained handling. The ACO algorithm within MIDACO is based on so called multi-kernel gaussian probability density functions (PDF's), which generate samples of iterates (also called *Ants*). For integer decision variables, a discretized version of the PDF is applied (see [21]). Figure 1 illustrates a Gauss PDF with three individual kernel PDF's for a continuous and integer domain.

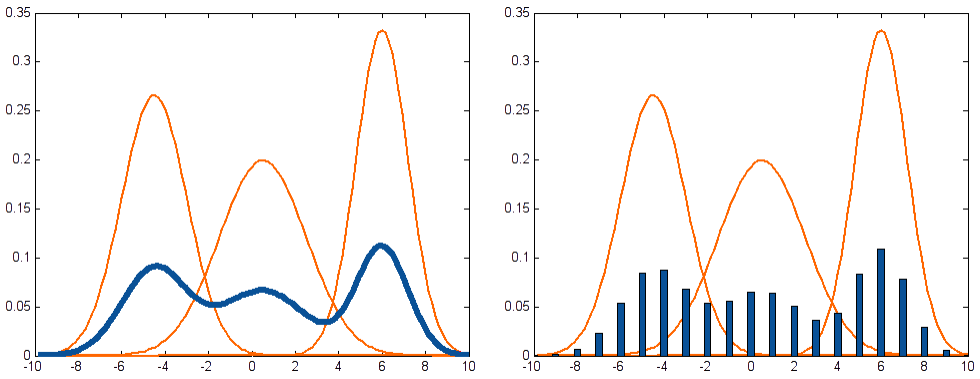


Figure 1: Continuous (left) and discretized (right) multi-kernel Gauss PDF

Constraints are handled within MIDACO by the Oracle Penalty Method which is an advanced method especially developed for heuristic search algorithms (like ACO, GA or PSO). This method aims on finding the global optimal solution by using a parameter called *Oracle* (or *Omega* in [23]), which corresponds directly to the objective function value $f(x)$. The method is self-adaptive and therefore MIDACO can also be classified as a self-adaptive algorithm. Figure 2 illustrates the shape of the extended oracle penalty function depending on the objective function value $f(x)$ and the *residual* value $res(x)$, which represents the constraint violation of $g(x)$ (measured in the commonly used L_1-norm).

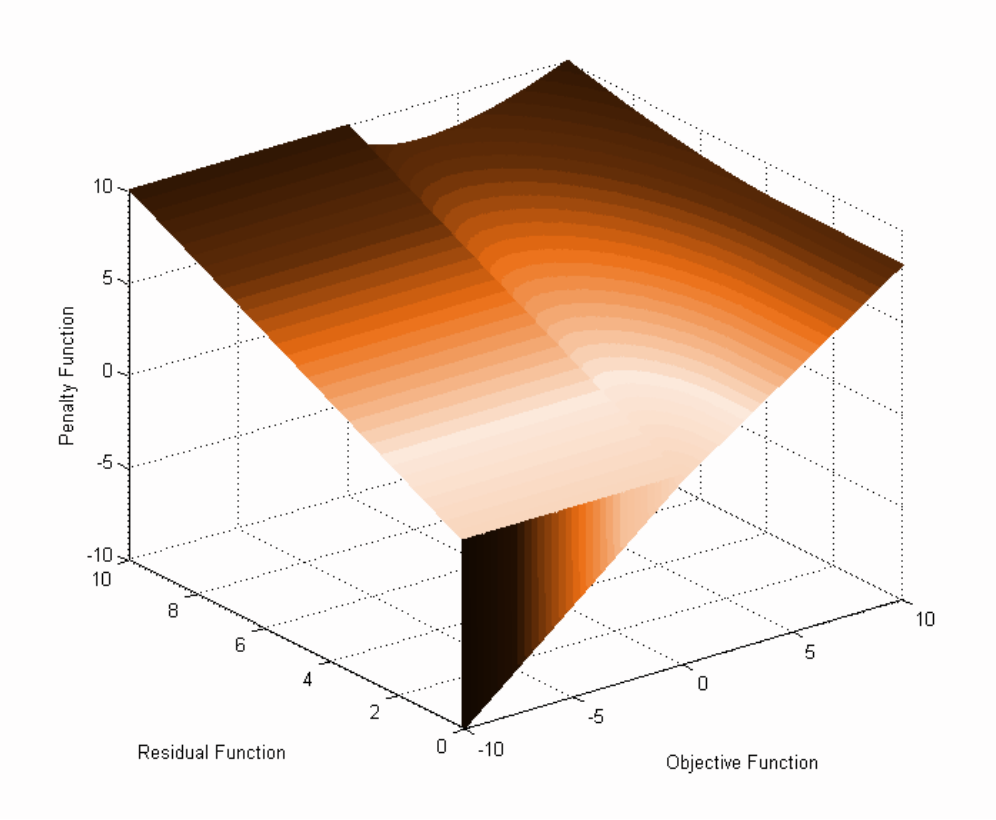


Figure 2: Shape of the extended oracle penalty function

Like the majority of evolutionary optimization algorithms, MIDACO considers the objective $f(x)$ and constraint $g(x)$ functions as **black-box** functions. This means that only the returning f and g values for some input vector x are recognized by MIDACO. No particular knowledge on how the objective and constraint function are calculated is required by MIDACO. Consequently the objective and constraint functions may exhibit any critical function property, like (strong) non-linearity, non-convexity, non-smoothness, non-differentiability, discontinuities and even stochastic noise.

In order to enhance its basic ACO framework, MIDACO implements many heuristics to escape from local optima and to improve the overall performance in reaching the global optimal solution. However, like all heuristic algorithms, MIDACO does not provide a guarantee for reaching the global optimal solution. The main motivation behind MIDACO is to provide a robust software tool that can optimize complex real world applications in a reasonable time to a reasonable good solution. Extensive numerical test show (see [24], [23] and especially [31]) that MIDACO is able to obtain global optimal solutions fast and reliable for a large set of MINLP benchmark problems. Numerical comparisons of MIDACO with established deterministic MINLP algorithms can be found for example in [24]. Numerical comparisons of MIDACO with stochastic search algorithms (including genetic algorithms, scatter search, variable neighbourhood and covariance matrix based search) can be found for example in [21], [22] or [27]. A collection of general global optimization problems (including well known NLP benchmarks like Rosenbrock, Schwefel or Rastrigin) that can be solved by MIDACO are available at the MIDACO benchmark website. Please note that the MIDACO runtimes and capabilities (e.g. number of variables and number of constraints) considered at the MIDACO benchmark website are at the state-of-the-art for evolutionary computing.

Because the class of MINLP problems cover purely continuous (NLP) as well as purely integer (IP) problems and the presence of constraints is an optional choice, MIDACO can be applied to a wide range of optimization problems. On the MIDACO applications website some examples of MIDACO utalization is presented for interplanetary space trajectory design [13], [25], [28], [29], construction and control of launch vehicles [26], operation of satellite constellations [32], low-thrust orbit transfer optimization [3], structural optimization of aircraft frames [34], attitude of quadrotors [14], chemical plant layout [22], [27], [12], control of cogeneration systems [17], waste water treatment [22], water supply networks [9], optimal camera placement [15], soil parameter optimization [33], meta-material fabric design [11], distance-to-default models in finance [2], [18], sales forecasting [10], wireless network telecommunication [1], [4], [5], [6], [7], [8], structural optimization of submarines [35] and parameter optimization in bio-technology [19].

For problems based on CPU-time expensive simulations (this means, the evaluation of the objective and/or constraint functions requires a significant amount of time), MIDACO offers an efficient parallelization strategy: MIDACO allows to evaluate the problem functions for several solution candidates in parallel. This strategy can significantly reduce the overall optimization time. The parallelization strategy in MIDACO is implemented by reverse communication which is a very robust and portable concept that can scale up to thousands of threads/cores. Due to this concept, MIDACO is able to offer its parallelization strategy in several programming languages for various parallelization schemes, including C/C++ (openMP, openMPI, GPGPU), Matlab (parfor), R (dopar), Java (Fork/Join) or Python (multiprocessing, mpi4py, pyspark). Easy to use example templates for MIDACO running with parallelization can be found on the MIDACO parallelization website.

The scope of this user manual is to provide practical guidelines on how to solve an optimization problem with the MIDACO software. Readers with a deeper interest in the theoretic details of the ACO algorithm within MIDACO can find more information in several publications (e.g. [21] or [22]), whereas [25] provides the most comprehensive and detailed explanation (including a simple step by step example). Detailed information on the development and properties of the oracle penalty method can be found in [23] or on the Oracle Penalty Method website.

This user manual is structured as follows: Section 1 describes how to present the optimization problem to MIDACO. Section 2 describes printing and output issues, particular the *screen* and *solution* file created by MIDACO. Section 3 explains the various stopping criteria of MIDACO and how they can be utilized and combined. In Section 4 the available parameters to tune the MIDACO performance or modify the MIDACO settings are described. Section 5 provides information on how to solve multi-objective problems with MIDACO. Section 6 discusses the parallelization mechanism of MIDACO. Section 7 presents some tips and tricks for advanced users who face challenging problems or seek some further details. Finally, Setion 8 presents a complete list of all IFLAG messages used by MIDACO.

This user manual assumes that the reader has already successfully downloaded and executed one of the small example problems that are distributed together with the limited MIDACO version, available here. Running these examples should be straight forward. However, if problems were experienced nevertheless, please consult the MIDACO troubleshooting website or contact the authors directly. Answers to specific questions can also be found on the MIDACO FAQ website.

1 Optimization Problem

This section explains how an optimization problem must be presented to MIDACO. The MIDACO algorithm considers an optimization problem in its most fundamental form: A black-box which returns the corresponding objective function values F(X) and constraint function values G(X) for some input variables X. This black-box concept is commonly used in evolutionary algorithms and gives the user complete freedom to define and calculate the objective and constraint values in whatever form is preferred, including calling complex subprograms in different programming languages. Due to this black-box concept it is furthermore allowed that the objective and constraint functions have critical properties (like discontinuity or stochastic noise) or that their actual mathematical formulation is truly unknown.

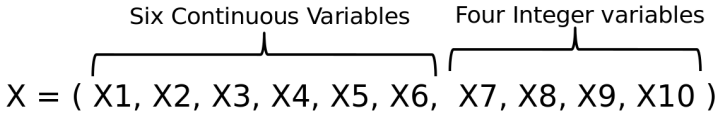
In case of mixed integer problems, where continuous and discrete (also called *integer*) variables are present simultaneously, the continuous variables are stored first in the vector of variables X, while the discrete ones are stored last in X. The distinction between equality and inequality constraints in the constraints vector G(X) is handled the same way: The equality constraints are stored first, the inequality constraints are stored last in the constraints vector G. As example, consider a constrained mixed integer problem with the following problem dimensions:

N	=	10	M	=	5
NI	=	4	ME	=	3

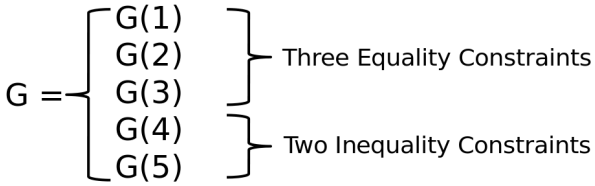
where:

N	:	Number of variables (in total)
NI	:	Number of integer variables
M	:	Number of constraints (in total)
ME	:	Number of equality constraints

then the distinction between continuous and integer variables in X is as follows:



and the distinction between equality and inequality constraints in G is as follows:



Some lower and upper bounds (XL and XU) for the decision variables X must be provided for any problem. A starting point X (also called initial solution or initial point) must be provided as well, however this can be any point (vector of decision variables X) that lies inbetween the bounds XL and XU. By default, the lower bounds are assumed as starting point in all example problems provided with MIDACO. In general it is recommended, to keep the search space (defined by XL and XU) as small as possible, as MIDACO will explore the entire search space (Hint: Use the BOUNDS-PROFIL to identify, where a reduction of the search space might be possible). In contrast to this, the starting point is normally not a critical issue for MIDACO.

In case the starting point X violates its lower or upper bound, MIDACO will raise the error message IFLAG=204 or IFLAG=205 respectively. In case integer variables the corresponding lower and upper bound value must also be a discrete value. In case the starting point X has some variables declared as integer type but those variables have a continuous value (e.g. 0.123), MIDACO will raise error message IFLAG=881. In case the bound value for an integer variable is a continuous value (e.g. 0.123) MIDACO will raise error message IFLAG=882 or IFLAG=883.

Note that MIDACO will always respect the integer type of variables. This means that MIDACO will never try to submit a continuous value for an integer type declared X variable for evaluation to the problem function. This is an important feature that distinguishes the MIDACO algorithm from classical algorithms for MINLP (like Branch & Bound) which normally require the *relaxation* of integer variables, which is a temporary violation of their integer type into the continuous domain.

1.1 Problem Dimensions, Bounds and Starting Point

This subsection illustrates how to declare the dimensions of the optimization problem, the lower and upper bounds and the starting point for the decision variable vector X. The problem dimensions refer there to the size of the F, G and X arrays. Below Matlab screenshot from the *example_MINLPc.m* illustrates a problem setup with 1 objective function and 4 variables, two of them of discrete type. It furthermore considers 3 constraints, one of them of equality type:

```
% STEP 1.A: Problem dimensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.o  = 1; % Number of objectives
problem.n  = 4; % Number of variables (in total)
problem.ni = 2; % Number of integer variables (0 <= nint <= n)
problem.m  = 3; % Number of constraints (in total)
problem.me = 1; % Number of equality constraints (0 <= me <= m)

% STEP 1.B: Lower and upper bounds 'xl' & 'xu'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.xl = [ 1, 1, 1, 1 ];
problem.xu = [ 4, 4, 4, 4 ];;

% STEP 1.C: Starting point 'x'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.x  = problem.xl; % Here for example: 'x' = lower bounds 'xl'
```

1.2 Problem Function Call

This subsection discusses the software function call to the optimization problem. As explained above, the only three elements that need to be communicated between MIDACO and the optimization problem are the vector of decision variables X, the vector of objective functions F(X) and the vector of constraint values G(X). In the example problems distributed on the MIDACO website, those problem function calls are given by their programming language as follows:

Matlab	:	[f, g] = problem_function(x)
Python	:	problem_function(x) (return f, g)
C/C++	:	problem_function(*double* *f, *double* *g, *double* *x)
Fortran	:	PROBLEM_FUNCTION(F,G,X)
R	:	problem_function <- function(f,g,x)
C#	:	blackbox(double[ ] f, double[ ] g, double[ ] x)
Java	:	blackbox(double[ ] f, double[ ] g, double[ ] x)

Below Matlab screenshot from the *example_MINLPc.m* illustrates the problem_function setup, in which for the input vector of decision variables x some calculation is performed to generate a single objective value f(1) and three constraint function values g(1), g(2) and g(3).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%   OPTIMIZATION PROBLEM   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ f, g ] = problem_function( x )

% Objective functions F(X)
  f(1) = (x(1)-1)^2 ...
       + (x(2)-2)^2 ...
       + (x(3)-3)^2 ...
       + (x(4)-4)^2 ...
       + 1.23456789;

% Equality constraints G(X) = 0 MUST COME FIRST in g(1:me)
  g(1) = x(1) - 1;
% Inequality constraints G(X) >= 0 MUST COME SECOND in g(me+1:m)
  g(2) = x(2) - 1.333333333;
  g(3) = x(3) - 2.666666666;

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%   END OF FILE   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Note that the vector index i for the F, G and X arrays may start with $i = 1$ in some languages (e.g. Matlab, R, Fortran) while it starts with $i = 0$ in other languages (e.g. C++, Python, Java).

1.3 Passing Additional Input/Output Arguments

The problem function call provided in the examples is reduced to the bare minimum, that is communication only the F,G and X vectors. In case the user wishes to change the name of the problem function and/or pass additional input/output arguments, the problem function call can freely be changed to whatever layout is desired. In case of C/C++ and Fortran such changes can be done directly in the example file source code. In case of other languages (like Matlab or Python) the source code in the language specific gateway file must be changed. Changing the name and/or arguments of the problem function can be done easily with a little programming effort by the user.

Below is a Matlab pseudo-code of how a modified problem function call may look like:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%   OPTIMIZATION PROBLEM   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ f, g, EXTRA_OUTPUT ] = USER_MODEL_NAME( x, EXTRA_INPUT )

###### Do something with EXTRA_INPUT ######

% Objective functions F(X)
  f(1) = (x(1)-1)^2 ...
       + (x(2)-2)^2 ...
       + (x(3)-3)^2 ...
       + (x(4)-4)^2 ...
       + 1.23456789;

% Equality constraints G(X) = 0 MUST COME FIRST in g(1:me)
  g(1) = x(1) - 1;
% Inequality constraints G(X) >= 0 MUST COME SECOND in g(me+1:m)
  g(2) = x(2) - 1.333333333;
  g(3) = x(3) - 2.666666666;

###### Prepare EXTRA_OUTPUT ######

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%   END OF FILE   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

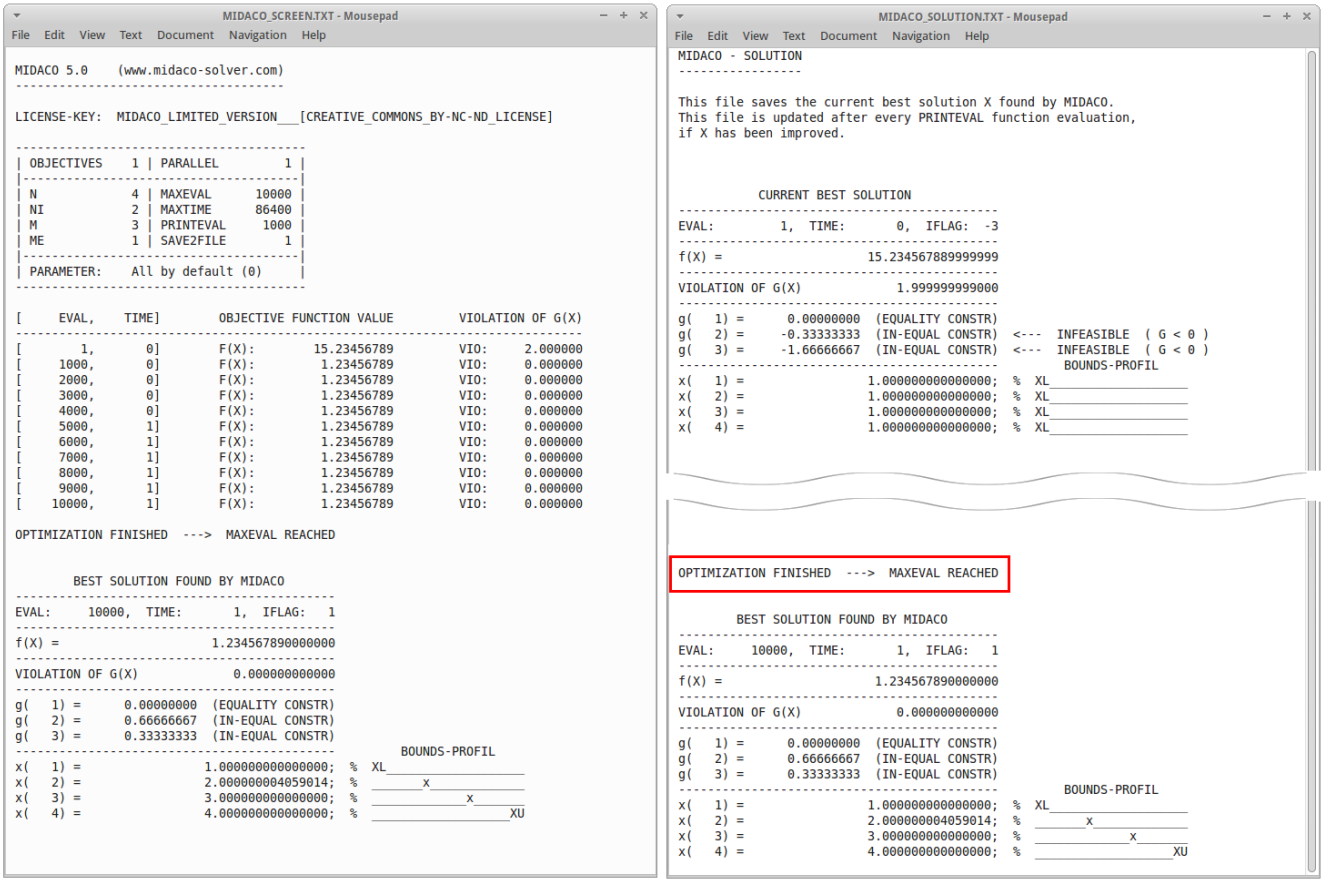
1.4 Verifying a Problem Implementation

If MIDACO (or any other optimizer) should be used to solve a specific problem, it is crucial that the problem is implemented correctly, otherwise a GIGO scenario might occur. In order to verify a problem implementation, it is recommended that before the actual optimization begins the user executes a single function evaluation. This can be achieved by setting the option MAXEVAL = 1 (see Section 3). By setting MAXEVAL=1 only the starting point will be evaluated and MIDACO stops immediately, reporting the corresponding objective and constraint values. The gives the user the chance to manually check if all reported objective and constraint values are reasonable.

2 MIDACO Screen and Solution

If printing is enabled, MIDACO produces two output text files:

MIDACO_SCREEN.TXT and MIDACO_SOLUTION.TXT



The MIDACO screen file is identical to the output displayed on the console/comand window. The MIDACO screen and solution file layout is basically identical in all programming languages. Depending different langauges, minor differences only apply to the vector index of F,G, and X and the comment symbol used for the bounds profiler. All abbreviations used in the MIDACO screen and solution file are explained in Table 1.

Table 1: Abbreviations used in the MIDACO screen and solution output files (Figure 2)

OBJECTIVES	Number of objective functions
PARALLEL	Number of parallel processed problem function calls (also called *co-evaluation*)
N	Number of variables in total
NI	Number of integer variables $0 \leq$ NI $\leq$ N
M	Number of constraints in total
ME	Number of equality constraints $0 \leq$ ME $\leq$ M
MAXEVAL	Maximum number of function evaluation (stopping criteria, see Section 3)
MAXTIME	Maximum CPU-time budget for execution (stopping criteria, see Section 3)
PRINTEVAL	Print frequency of the current best solution
SAVE2FILE	Create text-file output [0=No, 1=Yes, 2=Yes + create history file]
PARAM	Parameter for MIDACO tuning (default = 0, see Section 4)
EVAL	Number of performed function evaluation
TIME	Number of performed CPU-time Seconds
F(X)	Current best objective function value, found after EVAL evaluation
VIOLATION	Violation of constraints: measured as L1-Norm (Wikipedia) over vector G
IFLAG	Information flag used by MIDACO to indicate final status, warnings or errors
F(i)	Numerical value for individual objective F_i
G(i)	Numerical value for individual constraint G_i
X(i)	Numerical value for individual solution variable X_i

The BOUNDS-PROFIL is a graphical (ASCII) illustration of the relative position of X(i) regarding its lower (XL(i)) and upper (XU(i)) bound. If X(i) is closer than 0.1% to the lower or upper bound, the BOUNDS-PROFIL entry will display an upper-case 'XL' or 'XU' respectively, otherwise a lower-case 'x' is displayed.

The solution file contains the numerical values of the solution X for every iteration line printed on the screen. This means, the solution file is constantly updated after every PRINTEVAL function evaluation. This is an important feature as it gives the user the chance to access the full solution already during runtime and adds security in case an optimization run gets interrupted for some reason (e.g. server reboot or electricity black-out). Additionally, the very first solution (also called starting point, EVAL=1) and the final solution are displayed. The solutions are stored one after another. The BOUNDS-PROFIL is displayed for every solution stored in the solution file. All objectives F(i) and constraints G(i) are displayed individually. If a constraint is infeasible, it is highlighted by 'INFEASIBLE (G<0)' (for inequality constraints) or 'INFEASIBLE (G NOT=0)' (for equality constraints).

Note that X in the solution file is not updated, if X has not improved between two printing iterations. This is done to avoid unnecessary size enlargement of the solution file.

2.1 PRINTEVAL and SAVE2FILE

PRINTEVAL is the critical parameter to control how often the current best solution is printed on the screen. Note that this parameter is completely independent from any algorithmic iteration within MIDACO. Therefore the user can freely set PRINTEVAL in such a way, that the output frequency is convenient for display. Small values (e.g. 10, 123, 500) for PRINTEVAL will result in a faster output frequency. Large values (e.g. 10000, 100000 or 1000000) will result in a slower output frequency. The fastest possible output frequency is given by PRINTEVAL = 1, which means that after every evaluation the current best solution found by MIDACO is displayed. This option is only useful for very time intensive problems, or for debugging purposes. In general it is recommended to set large values for PRINTEVAL. This way the user gets a better overview on the optimization progress and MIDACO runs a little bit faster (because the printing commands needs less often to be executed). For most real-world applications it is sufficient enough to set PRINTEVAL in such way that a new printout line happens only every couple of minutes.

The creation of the output files is optional. If SAVE2FILE is set to zero, no output file will be created. If no output at all is desired (for example if MIDACO should be silently embedded within a high-level software and only the final solution vector X should be further numerically processed in such high-level software), all visual output can be suppressed by setting PRINTEVAL to zero.

Thus: setting PRINTEVAL=0 and SAVE2FILE=0 completely silents MIDACO.

2.2 Solution History File

Additionally to the screen and solution file, MIDACO can produce a complete history of all evaluated iterates X. In order to enable MIDACO creating a history file, set the option SAVE2FILE equal to two. If the option SAVE2FILE=2 is set, MIDACO will automatically create a file named "MIDACO_HISTORY.TXT" which contains all evaluated iterates X together with their corresponding objective values F and constraint values G.

Creating a history file can be useful for applications which are CPU-time intensive and were full access to all available evaluation results is desired. Especially if parallelization is applied, the creation of a history file offers a simple way to keep track on all processed evaluations. Note that creating a history file for CPU-time cheap problems (e.g. benchmarks) with millions of evaluations can easily lead to history files of huge size (several Gigabytes). User caution is therefore advised.

The solution format which is used in the history file is identical to the format used for the pareto front file "MIDACO_PARETOFRONT.TXT" created for multi-objective problems. Therefore the contend of the history file can also be plotted with the PlotTool (see Section 5.2) or further processed as ".csv" file (for example in Excel).

For further hints on using the history file see Section 7.7, 7.8 and 7.9.

3 MIDACO Stopping Criteria

The stopping criteria for MIDACO can be categorized into two groups: Hard limit criteria and algorithmic criteria. Hard limit criteria are MAXTIME and MAXEVAL which cause MIDACO to stop its optimization process based on a maximal budget of CPU-time or the number of function evaluation. Algorithmic criteria are FSTOP, ALGOSTOP and EVALSTOP which cause MIDACO to stop based on some algorithmic decision. All stopping criteria can be freely combined by the user to fit a specific purpose at hand. The following sub-sections illustrate each criteria in detail and furthermore give some example setup scenarios.

3.1 Hard Limit Criteria

Here stopping criteria are discussed which are based on a hard limit, such as time or evaluation.

3.1.1 MAXTIME

The MAXTIME criteria defines a maximal CPU-time budget measured in seconds. Freely set this stopping criteria to any value. Setting a very large value (like 1000000) practically disables this criteria. Most example problems provided with MIDACO use a dummy value of one day, which is 60*60*24. For quick orientation, below table displays usual time scales measured in seconds.

Minute	:	60	=	60
15 Minutes	:	60*17	$<\approx$	1000
Hour	:	60*60	=	3600
2.5 Hours	:	60*60*2.5	$<\approx$	10000
Day	:	60*60*24	=	86400
27 Hours	:	60*60*27	$<\approx$	100000
Week	:	60*60*24*7	=	604800

3.1.2 MAXEVAL

The MAXEVAL criteria defines a maximal budget of problem function evaluation. It is a distinctive feature of the MIDACO software implemenation to be able to stop exactly after any given number of evaluation (e.g. 123456). The user can therefore freely choose any arbitrary integer value for MAXEVAL.

MIDACO can quickly process millions of function evaluation within seconds, if the actual function evaluation is computationally cheap (like for benchmark problems). This means that for fast calculating applications, evaluations limits of 10000000 (ten million) or 100000000 (hundred million) are often reached within minutes. For those fast calculating applications it can be desirable to completely switch of the MAXEVAL stopping criteria. Therefore the MAXEVAL criteria will apply only for values lower than 999999999 (*"nine times nine"*). If the MAXEVAL stopping criteria is assigned any value greater or equal to 999999999, it will switch itself off completely.

Note that in contrast to above special scenario the very important case of CPU-time expensive applications, where only a few thousands or just hundreds of evaluation can be calculated within reasonable time, is discussed separately in Section 6.

3.2 Algorithmic Criteria

Here stopping criteria are discussed which are based on an algorithmic decision.

3.2.1 FSTOP

The FSTOP parameter is enabled if any value except exactly zero (0.0E+0) is assigned to it. If MIDACO reaches a feasible solution with an objective function value lower or equal to FSTOP, MIDACO will stop. Note that this stopping criteria refers the first objective function in case of multi-objective problems. It is important to note that MIDACO will be strict about the FSTOP value, therefore MIDACO does not add any tolerance to the FSTOP value. If zero is the desired value for FSTOP, some dummy value like 0.000000001 can be used as FSTOP.

3.2.2 ALGOSTOP

The ALGOSTOP parameter is enabled if any positive integer value (e.g. 1,2,3,...) is assigned to it. This criteria will measure the algorithmic improvement between MIDACO internal ACO restarts. The value of ALGOSTOP defines the maximal number of consecutive MIDACO internal ACO restarts without improvement of the (feasible) objective function value. For example: If ALGOSTOP=10 is set, than MIDACO will perform its optimization search until 10 consecutive internal ACO restarts did not further improve the current solution.

The higher the value for ALGOSTOP is set (like 10, 50, 100 or higher) the higher the chance that MIDACO reached the global optimal solution. In such regard this stopping criteria is the most advanced to indicate global optimality. The significant drawback of this stopping criteria is that it might require many (normally thousand or even millions) of function evaluation. It is therefore only suitable for applications which are CPU-time cheap to evaluate. When experimenting with the ALGOSTOP criteria, values such as 1, 5, 10 or 30 might be used at first to get a feeling for the run-time effect on a specific application.

For applications with CPU-time expensive evaluation the EVALSTOP criteria is more appropriate.

3.2.3 EVALSTOP

The EVALSTOP parameter is enabled if any positive integer value (e.g. 1,2,3,...) is assigned to it. It works similar to the ALGOSTOP criteria but with the significant difference that it does not consider complete MIDACO internal ACO restarts but individual function evaluation. For example: If EVALSTOP=999 is set, than MIDACO will perform its optimization search until 999 consecutive function evaluation did not further improve the current solution.

The lower the value for EVALSTOP is set (like 1000, 100 or lower) the faster MIDACO will stop. In case EVALSTOP=1 is set, MIDACO will stop immediately after any function evaluation which did not improve the current solution. Therefore for very small EVALSTOP values (like 1,2,3,...) MIDACO will stop very fast. Goal of this stopping criteria is to provide an algorithmic stopping criteria that is not as expensive as ALGOSTOP in the number of required function evaluation, but that is still based on an algorithmic measure. When experimenting with the EVALSTOP criteria, values such as 10000, 1000 or 500 might be used at first to get a feeling for the run-time effect on a specific application.

The EVALSTOP criteria can further be fine-tuned by specifying the precision (in relative percentage) applied to measure if a new solution is considered as improvement or not. The default precision for EVALSTOP is 0.001, which is 0.1% in relative percentage. In case a different precision should be used, such precision should be appended as floating point extension to the EVALSTOP value. For example: MIDACO should stop after 333 consecutive function evaluation without improvement of 0.25% relative percentage of the objective function value. Then setting EVALSTOP = 333.0025 will enable such stopping criteria. If no specific floating point extension is given to EVALSTOP, the default precision of 0.001 is applied automatically.

3.3 Example Scenarios

Here some example scenarios are given how to set up a single or several stopping criteria together.

3.3.1 Single Evaluation

If MAXEVAL=1 is set, MIDACO will only perform a single function evaluation. This is by definition the starting point X provided by the user. This scenario is useful to verify a problem implementation, as it give the user the chance to check in detail all objective and constraint function values reported for the starting point X to be reasonable. This option is also useful to re-produce a given solution (thus re-evaluating it).

3.3.2 CPU-Time expensive application

This scenario exemplifies a CPU-time expensive application where only a low number of function evaluation are available (for example a complex machine simulation model). A suitable setup for such application might look something like this:

MAXTIME	=	50000
MAXEVAL	=	999999999 ($\rightarrow$ disabled)
FSTOP	=	0 ($\rightarrow$ disabled)
ALGOSTOP	=	0 ($\rightarrow$ disabled)
EVALSTOP	=	50

Above setup assigns a hard time limit of 50000 seconds (about half a day) and further addresses an EVALSTOP=50 stopping criteria, in the hope that such stopping criteria is reached before the actual time limit is reached. All other criteria are disabled.

3.3.3 CPU-Time cheap application

This scenario exemplifies a CPU-time cheap application where a high number of function evaluation can quickly be calculated (like in academic benchmark problems). A suitable setup for such application might look like something like this:

MAXTIME	=	60*60*24
MAXEVAL	=	10000000
FSTOP	=	0.00000001
ALGOSTOP	=	200
EVALSTOP	=	0 ($\rightarrow$ disabled)

Above setup assigns a hard function evaluation limit of 10000000 (ten million) and further addresses an FSTOP=0.00000001 and ALGOSTOP=500 criteria. Thus, MIDACO will stop if a (feasible) solution with objective lower or equal 0.00000001 is found, or the MIDACO internal ACO performed 200 consecutive restarts without further solution improvement or the evaluation budget is spent. This setup is not concerned with the actual time and thus practically disables the MAXTIME criteria by given it a full day.

3.3.4 Infinite Run

The MIDACO software is constructed in such way that it is able to potentially run forever, except the text file output it will not accumulate any data or memory and no internal algorithmic element will run against some bound and crash.

A setup where MIDACO is practically running forever can be achieved by setting MAXEVAL and MAXTIME to 999999999 while keeping FSTOP, ALGOSTOP and EVALSTOP by their default value (zero). Such setup might appear absurd at first, but is commonly used in practice. By disabling all automatic stopping criteria for MIDACO the user takes the final decision when to stop the optimization run (e.g. by shutting down the program/computer) in his/her own hand, giving MIDACO the highest chance to find the global optimal solution (or best spread of pareto points).

Note that the MIDACO_SOLUTION.TXT and MIDACO_PARETOFRONT.TXT files are updated with the latest solution(s) at each PRINTEVAL function evaluation. Therefore, even under an infinite run scenario the user has always access to the solutions and can already further process them or plot the pareto front while the actual MIDACO optimization run is still ongoing.

4 MIDACO Parameter

MIDACO offers twelve parameters to customize its performance and behaviour. The individual parameters are explained in the following subsections. The default value for all parameter is zero.

4.1 PARAM(1) : ACCURACY

This parameter defines the accuracy tolerance for the constraint violation. MIDACO considers an equality constraint to be feasible, if $|G(X)| \leq$ PARAM(1). An inequality is considered feasible, if $G(X) \geq$ -PARAM(1). If the user sets PARAM(1) = 0, MIDACO uses a default accuracy of 0.001. This parameter has strong influence on the MIDACO performance on constraint problems. For problems with many or difficult constraints, it is recommended to start with some test runs using a less precise accuracy (e.g. PARAM(1)=0.1 or PARAM(1)=0.05) and to apply some refinement runs with a higher precision afterwards (e.g. PARAM(1)=0.0001 or PARAM(1)=0.0000001).

Note that the displayed "VIOLATION OF G(X)" (see MIDACO screen) expresses the L1-Norm over the vector G in respect to PARAM(1). In case all constraints are feasible to to accuracy defined by PARAM(1), the "VIOLATION OF G(X)" is displayed as zero.

4.2 PARAM(2) : SEED

This parameter defines the initial seed for MIDACO's internal pseudo random number generator. The seed determines the sequence of pseudo random numbers sampled by the generator. Therefore changing this value will lead to different results by MIDACO. The seed must be an integer greater or equal to zero (e.g. PARAM(2) = 0,1,2,3,...1000).

Note that MIDACO runs are 100% reproducible, if performed with the same seed (and executed on the same machine with identical compiler settings). The advantage of a user specified random seed is, that promising runs can easily be reproduced. This is in esp. useful, if a run was unintentionally interrupted and should be restarted again.

For difficult problems it can be useful to execute several runs of MIDACO with different random seed, rather than performing only one very long run.

4.3 PARAM(3) : FSTOP

This parameter enables a stopping criteria for MIDACO. The FSTOP stopping criteria is based on an objective function value to be reached. Full details on the FSTOP parameter are described in Section 3.2.1. For multi-objective problems the FSTOP values applies for the first objective.

4.4 PARAM(4) : ALGOSTOP

This parameter enables a stopping criteria for MIDACO. The ALGOSTOP stopping criteria is based on the algorithmic process of MIDACO. Full details on the ALGOSTOP parameter are

described in Section 3.2.2.

4.5 PARAM(5) : EVALSTOP

This parameter enables a stopping criteria for MIDACO. The EVALSTOP stopping criteria is based on the algorithmic process of MIDACO taking account the number of function evaluation. Full details on the EVALSTOP parameter are described in Section 3.2.3.

4.6 PARAM(6) : FOCUS

This parameter forces MIDACO to focus its search process around the current best solution. This parameter is probably the most powerful and widely applicable one. For many problems, tuning this parameter is useful and will result in a faster convergence speed (in esp. for convex and semi-convex problems). This parameter is also in especially useful for refining solutions. If PARAM(6) is not equal zero, MIDACO will apply an upper bound for the standard deviation of its Gauss PDF's (see Section , Figure 1). The upper bound for the standard deviation for continuous variables is given by (XU(i)-XL(i))/FOCUS, whereas the upper bound for the standard deviation for integer variables is given by MAX((XU(i)-XL(i))/FOCUS,1/SQRT(FOCUS)).

In other words: The larger the value of FOCUS, the closer MIDACO will concentrate its search around its current best solution.

The value for PARAM(6) must be an integer. Smaller values for FOCUS (e.g. 10 or 100) are recommend for first test runs (without a specific starting point). Larger values for FOCUS (e.g. 10000 or 100000) are normally only useful for refinement runs (where a specific solution is used as starting point).

Furthermore it is possible to submit negative values for FOCUS (e.g. -1000 or -10000). In such case, the minus ("-") is not treated numerically; instead, MIDACO will interpret the minus ("-") as an information flag. While for positive FOCUS values MIDACO will also explore other regions of the search space by independent restarts, a negative FOCUS value disables the independent restart option within MIDACO. In other words: For a negative FOCUS value MIDACO is focused entirely on the starting point. Therefore negative FOCUS values should be used only for refinement runs, where the user has high confidence in the quality of the specific solution used as starting point.

4.7 PARAM(7) : ANTS

This parameter allows the user to fix the number of *ants* (iterates) which MIDACO generates within one generation (major iteration of the evolutionary ACO algorithm). This parameter must be used in combination with PARAM(8). Using the ANTS and KERNEL parameters can be promising for some problems (in esp. large scale problems or CPU-time intensive applications). However, tuning these parameters might also significantly reduce the MIDACO performance. If PARAM(7) is equal to zero, MIDACO will dynamically change the number of ants per generation. See PARAM(8) for more information on handling this parameter.

4.8 PARAM(8) : KERNEL

This parameter allows the user to fix the number of kernels within MIDACO's multi-kernel Gauss PDF's (see Section , Figure 1). The kernel size corresponds also to the number of solutions stored in MIDACO's solution archive. On rather convex problems it can be observed, that a lower kernel number will result in faster convergence while a larger kernel number will result in lower convergence. On the contrary, a lower kernel number will increase the risk of MIDACO getting stuck in a local optimum, while a larger kernel number increases the chance of reaching the global optimum. The kernel parameter must be used in combination with the ants parameter. In Table 2 some examples of possible ants/kernel settings are given and explained below.

Table 2: Example settings for ANTS/KERNEL combinations

Setting 1		Setting 2		Setting 3		Setting 4	
ANTS	2	ANTS	30	ANTS	500	ANTS	100
KERNEL	2	KERNEL	5	KERNEL	10	KERNEL	50

The 1st setting is the smallest possible one. This setting might be useful for very CPU-time expensive problems where only some hundreds of function evaluation are possible or for problems with a specific structure (e.g. convexity). The 2nd setting might also be used for CPU-time expensive problems, as a relatively low number of ANTS is considered. The 3rd and 4th setting would only be promising for problems, with a fast evaluation time. As tuning the the ants and kernel parameters is highly problem depended, the user needs to experiment with those values.

Note that the maximum kernel number for MIDACO is fixed to 100.

4.9 PARAM(9) : ORACLE

This parameter specifies a user given oracle parameter to the penalty function within MIDACO. This parameter is only relevant for constrained problems. If PARAM(5) is not equal to zero, MIDACO will use PARAM(5) as initial oracle (otherwise MIDACO will use 10^9 as initial oracle). This option can be especially useful for constrained problems where some background knowledge on the problem exists. For example: It is known that a given application has a feasible solution X corresponding to F(X)=1000 (e.g. plant operating cost in Dollar). It might be therefore reasonable to submit an oracle value of 800 or 600 to MIDACO, as this cost region might hold a new feasible solution (to operate the plant at this cost value). Whereas an oracle value of more than 1000 would be uninteresting to the user, while a too low value (e.g. 200) would be unreasonable. More information on the oracle penalty method can be found at the Oracle Penalty Method website.

4.10 PARAM(10) : PARETOMAX

This parameter defines the maximal number of non-dominated solutions (also called *pareto points*) stored by MIDACO. The default value used by MIDACO is 100 (if PARAM(10)=0 is set). User can freely set any arbitrary large integer for PARETOMAX, for example PARAM(10)=1000 or PARAM(10)=100000. Note that larger PARETOMAX values will require more memory and will normally slow slow down the internal calculation time of MIDACO, due to increased pareto-filtering efforts. For many applications a PARETOMAX value <= 1000 is sufficient. The user can also specify smaller values, for example PARAM(10) = 30, which will normally speed up the internal calculation time of MIDACO.

Be default, MIDACO will filter all objective functions for dominance. A special case may however occur if the first objective function should be excluded from the dominance filter criteria. This might for example be the case if the first objective is a complex construction of different objectives (including dynamically changing weights). In such cases, including the first objective in the filter criteria might lead to more but undesired pareto solutions. In order to exclude the first objective from the dominance filter, the PARETOMAX value should be flagged with a "-" (negative) sign. For example PARAM(10) = -500 will enable MIDACO to store up to 500 pareto points, excluding the first objective from the filtering criteria.

It is important to note that even if the first objective is excluded from the filtering criteria, its value is reported like normal in all solution displays and the MIDACO_PAERTOFRONT.TXT file. Thus, if an artificial construction is used as first objective (see e.g. Section 5.1) it is normally recommended to exclude the first objective from the pareto filtering criteria.

4.11 PARAM(11) : EPSILON

This parameter defines the precision used by MIDACO for its multi-objective dominance filter. If PARAM(11)=0 is set, the default value of EPSILON=0.001 is used. The value of 0.001 means that a new non-dominated solution is introduced in MIDACO's pareto point archive only if at least one of their objectives is at least 0.1% (that is 0.001 in numerics) better than the corresponding objective of any previous solution. The EPSILON value can have a great influence in the amount of pareto points found and MIDACO's internal calculation time.

For most applications, a value of EPSILON larger or equal to 0.001 is sufficient. Smaller values, such as PARAM(11)=0.00001 or PARAM(11)=0.00000001 will normally result in many (!) more pareto points reported. However, those pareto points are only slightly different from each other and might therefore not provide much useful information. A special case in multi-objective optimization are many-objective problems, which consider four or more objective functions. Those problems often easily generate many (thousands) of non-dominated solutions. For many-objective problems it can be useful to assign a larger EPSILON value, such as PARAM(11)=0.01 or PARAM(11)=0.1. Using such high EPSILON value will force MIDACO to store and report only pareto points with a significant difference in at least one of their objectives. Note that using larger EPSILON values will also greatly speed-up MIDACO's internal calculation times.

4.12 PARAM(12) : CHARACTER

The character parameter allows to activate MIDACO internal parameter settings. MIDACO 5.0 offers the following three pre-defined characters:

CHARACTER = 1	:	MIDACO internal parameters for continuous problem types
CHARACTER = 2	:	MIDACO internal parameters for combinatorial problem types
CHARACTER = 3	:	MIDACO internal parameters for *All-Different* problem types

If PARAM(12)=0 is set, MIDACO will decide by itself if the internal parameters for the continuous or combinatorial problem is choosen. The internal parameters for continuous problem types will enable a more fine-grained search process, while the internal parameters for combinatorial problem types will enable a more coarse-grained search process. A special case are *All-Different* problem types. Those problems require that all integer variables must contain a different value. A famous example for *All-Different* problems is the traveling salesman problem (TSP). In case of *All-Different* problems the CHARACTER=3 should be enabled. If CHARACTER=3 is set, MIDACO will generate only solutions which automatically satisfy the *All-Different* constraint. This means the *All-Different* constraint does not need to be explicitly formulated and provided by the vector of constrains G(X). MIDACO will take care of it automatically. When using the *All-Different* character it is to note that the staring point X must already satisfy the *All-Different* constraint, otherwise an IFLAG=402 error is raised.

Note that MIDACO's *All-Different* character can also be used for mixed integer problems. In such case the all-different constraint affects all integer variables, but does not affect any of the continuous variables.

5 Multi-Objective Optimization

Note that the new MIDACO 6.0 does not require the use of an overall objective function and can handle multiple objective functions fully automatically. See the MIDACO 6.0 release notes for a detailed explanation of the new ***balance parameter*** for multi-objective optimization.

MIDACO can handle optimization problems with a single objective as well as with multiple objective functions. In case of more than one objective function (that is $O \geq 2$), the vector of objective functions F(X) is expected to contain all individual objectives. MIDACO will concentrate its minimization effort exclusively to the first objective function value F_1, which is considered to be some kind of **overall objective function**. While the remaining individual $F_2, F_3, ..., F_O$ objective values are not directly subject to minimization, they are still considered and actively filtered by MIDACO for the non-dominance criteria (subject to minimization). The resulting set of non-dominated solutions (also called *pareto-front*) is reported by MIDACO in form of a text file output, named MIDACO_PARETOFRONT.TXT. Figure 3 illustrates the pareto front approximation obtained by MIDACO for ESA's Cassini1 benchmark problem [13], where the propulsion (calculated as the sum of launch deltaV and flight deltaV) is used as overall objective function.

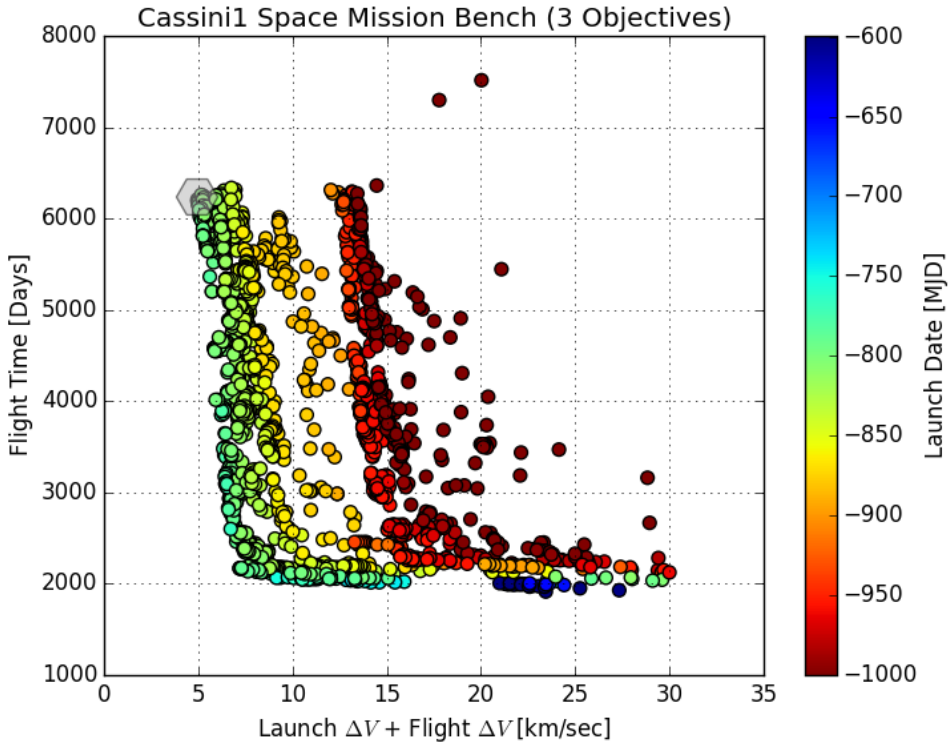


Figure 3: Pareto front approximation of ESA's Cassini1 benchmark.

While it may happen in some multi-objective real-world application that just single objective value should be optimized and the remaining ones should be purely monitored; the regular case is that several individual objectives should be optimized at the same time. In such case the construction of an overall objective function based on the individual objective functions is crucial. Therefore this section presents some common approaches to construct such overall objective function in sub-section 5.1.

Graphically illustrating the obtained set of non-dominated solutions (pareto-front) is an essential part in multi-objective optimization. Sub-section 5.2 gives an introduction to the *PlotTool.exe* which is a stand-alone graphic program that can be used to illustrate the pareto-front file reported by MIDACO.

Note that the MIDACO parameters PARETOMAX (Section 4.10) and EPSILON (Section 4.11) are relevant to the size and accuracy of the reported pareto-front as well as MIDACO's runtime performance on multi-objective problems.

5.1 Constructing an overall target function

This sub-section illustrates some possible examples on how an overall target function can be constructed out of multiple individual objective functions. As general example used in the following sub-sections, consider the optimization of a machine model with following four individual objectives:

Table 3: Example objectives of a machine model

$\mathcal{F}_{perform}$	=	Performance of the machine	(maximize)
$\mathcal{F}_{power}$	=	Power consumption of the machine	(minimize)
$\mathcal{F}_{cost}$	=	Cost to build the machine	(minimize)
$\mathcal{F}_{noise}$	=	Noise of the machine	(not specified)

5.1.1 Single Target

Picking a single objective out of several ones is the most elementary approach possible. This *single target* approach can be useful in real-world applications where the desired objective is very difficult (or time intensive) to optimize. While technically this approach does not differ from single-objective optimization, a multi-objective formulation with several objectives submitted to MIDACO can provide significant more information and benefit for the user. This approach is also useful if other objectives should only be monitored, without a preference for mini- or maximization.

As example, a single target objective approach to initially mentioned multi-objective machine model optimization might look like following, where F(1) is the overall target minimized by MIDACO:

F(1) =	- $\mathcal{F}_{perform}$
F(2) =	$\mathcal{F}_{power}$
F(3) =	$\mathcal{F}_{cost}$
F(4) =	$\mathcal{F}_{noise}$

In above example, the $\mathcal{F}_{perform}$ objective is the only single target to be maximized (by minimizing its negative transformation F(1) = - $\mathcal{F}_{perform}$), all other objectives are only monitored by MIDACO without preference for maxi- or minimization.

5.1.2 Weighted Sum

Creating a weighted sum of individual objective functions as overall objective is a very common approach to solve multi-objective problems. This approach requires that some information on the *scale* of the individual functions is known. Such information is used to *weight* the individual objectives in such way, that their impact on the overall target function is balanced equally or in a desired way. Note that weighted sums approaches can work well on convex (invert bent) pareto-fronts, however they do not work well on concave (outward bent) pareto-fronts.

As example, a weighted sum approach to initially mentioned multi-objective machine model optimization might look like following, where F(1) is the overall target minimized by MIDACO:

F(1) =	- $\frac{\mathcal{F}_{perform}}{10} + \frac{\mathcal{F}_{power}}{0.5} + \frac{\mathcal{F}_{cost}}{200}$
F(2) =	- $\mathcal{F}_{perform}$
F(3) =	$\mathcal{F}_{power}$
F(4) =	$\mathcal{F}_{cost}$
F(5) =	$\mathcal{F}_{noise}$

In above example, the overall target function F(1) is constructed using three out of four available objectives. Each individual objective in the F(1) function is weighted by some value that reflects the *scale* of the individual objective. Note that the total length of the F vector is 5, considering the four original individual objectives in F(2), F(3), F(4) and F(5). Further note that the F_{noise} was not included in the overall target objective as it should not have a preference for mini- or maximization, however it should still be monitored so that its behaviour can later on be displayed in the pareto-front pictures. Note that above values of {10, 0.5, 200} are only example dummy values for illustration.

5.1.3 Weighted Distance Function

Distance function approaches to multi-objective problems are a little more advanced than weighted sum approaches. This is because such approach requires that not only some knowledge of the

scale of the individual objectives is available, but also some knowledge on the best values of the individual objectives (sometimes also called *ideals* or *utopias*). Because knowledge on the *utopias* of the individual objectives directly implies knowledge on the *scale* of the individual objectives, it is useful to combine the distance function with the weighted sum approach. A great benefit of distance function based approaches is that those can work well on both: convex and concave pareto-fronts.

As example, a weighted distance function approach to initially mentioned multi-objective machine model optimization might look like following, where F(1) is the overall target minimized by MIDACO:

F(1) =	$-\frac{\lvert\mathcal{F}_{perform}-13\rvert}{10}+\frac{\lvert\mathcal{F}_{power}-0.666\rvert}{0.5}+\frac{\lvert\mathcal{F}_{cost}-166\rvert}{200}$
F(2) =	$-\mathcal{F}_{perform}$
F(3) =	$\mathcal{F}_{power}$
F(4) =	$\mathcal{F}_{cost}$
F(5) =	$\mathcal{F}_{noise}$

In above example the distance of each individual objective to its *ideal* value is measured as absolute distance. Note that for applying the weighted distance function approach it is not necessary to know the exact *ideal* value of each individual objective. A rough guess is normally sufficient. Note that above values of {13, 0666, 166, 10, 0.5, 200} are only example dummy values for illustration.

5.1.4 Additional Constraints

In addition to constructing and overall target function, it can be useful to introduce additional constraints on the individual objective functions. Adding those constraints can reduce the displayed pareto-front to an interesting region. If done properly, adding those constraints is easy and will not increase the complexity or behaviour of the optimization with MIDACO.

As example, consider the weighted distance function approach illustrated in section 5.1.3 and three additional constraints on the $\mathcal{F}_{cost}$ and $\mathcal{F}_{noise}$ objectives:

F(1) =	$-\frac{\lvert\mathcal{F}_{perform}-13\rvert}{10}+\frac{\lvert\mathcal{F}_{power}-0.666\rvert}{0.5}+\frac{\lvert\mathcal{F}_{cost}-166\rvert}{200}$
F(2) =	$-\mathcal{F}_{perform}$
F(3) =	$\mathcal{F}_{power}$
F(4) =	$\mathcal{F}_{cost}$
F(5) =	$\mathcal{F}_{noise}$
g(1) =	300 - $\mathcal{F}_{cost}$
g(2) =	$\mathcal{F}_{noise}$ - 50
g(3) =	100 - $\mathcal{F}_{noise}$

In above example the first additional constraint g(1) makes sure that only solutions with a cost value lower or equal to 300 are reported. The second and third constraint g(2) and g(3) further enforce that all reported solution have a noise value between 50 and 100. Adding such restrictions can reduce the displayed solution space to interesting regions and add clarity to the decision maker. Adding additional constraints is especially useful for problems with many individual objectives.

5.1.5 Auto-Scaling

While the weighted distance function approach described in Section 5.1.3 works well in practice on most applications its significant drawback is that the user needs to provide information on the *ideal* value of the individual objectives. Normally such knowledge is gained by experience and/or previous optimization attempts. If such knowledge is however not available or inaccurate, information on an advanced approach based on the recently introduced *Utopia-Nadir-Balance* [30] with fully automatic weight scaling is available at:

www.midaco-solver.com/index.php/more/multi-objective/autoscaling

5.2 PlotTool

Graphically illustrating the set of non-dominated solutions (also called pareto-front) is an essential part in multi-objective optimization. One possibility to illustrate the the pareto-front is using the PlotTool.exe [50MB] program, freely available here. The PlotTool allows to easily generate 2D and 3D illustrations of pareto-fronts stored in the regular MIDACO_PARETOFRONT.TXT file format.

The basic operations of the PlotTool should be self-explanatory. Users can zoom in and out by using the mouse-wheel for 2D graphics. For 3D graphics zoom in and zoom out is achieved by pressing the right mouse button and moving the mouse forward or backward. PlotTool will create a text file "PlotTool.def" when successfully creating a graphic. This PlotTool.def contains all available settings from the user form. User can manipulate the default entries of PlotTool by changing the entries in the "PlotTool.def" text file and restarting the PlotTool.exe. Except for illustrating the MIDACO_PARETOFRONT.TXT file the PlotTool can be used to convert the MIDACO_PARETOFRONT.TXT into a csv (comma separated format) file which can for example be opened via Excel. Note that the PlotTool accepts LaTeX formulas as input for figure title and legend descriptions. For example the title "`$\alpha-\beta-\gamma$ Design`" will be displayed as "$\alpha-\beta-\gamma$ Design". The PlotTool.exe will run on all 32 and 64 bit Windows platforms and can also be executed on Mac/Linux via emulators, such as Wine.

Note that the illustration of pareto front files created by MIDACO is in no way restricted to the PlotTool. Any kind of graphic capable program (like Matlab or gnuplot) can be used to illustrate the data stored in the pareto front file in whatever manner preferred.

Important note: Some anti-virus programs (e.g. SMADAV) might flag PlotTool.exe as potential risk. This is a false positive: The PlotTool.exe is based purely on Python and its matplotlib graphic library [16]. It does not contain any harmful software and will not upload/download any data.

6 Parallelization

MIDACO offers the possibility to evaluate multiple solution candidates in parallel. In the context evolutionary algorithms such feature is also known as *co-evaluation* or *fine-grained* parallelization. Figure 4 illustrates how a *block* of **P** solution candidates (x1, x2, x3, ... xP) is submitted for parallel evaluation and the corresponding objective and constraint values ([f1,g1],...,[fP.gP]) are returned to MIDACO.

If an optimization problem is *CPU-time expensive*, that means a single evaluation of the objectives and constraints requires a significant amount of time, parallelization is highly beneficial in reducing the overall time required to solve the problem. In a recent study (Schlueter & Munetomo [31]) it was numerically demonstrated on 200 benchmark problems that MIDACO's potential speed up by parallelization exhibits a nearly linear scale-up, which means it is most effective. For a parallelization factor of $P = 10$ the potential speed up was around 10 times, while for a parallelization factor of $P = 100$ the potential speed up was still around 70 times (see Figure 4 in [31]). Note that due to the parallelization overhead such speed ups are only expected if the time to calculate the objectives and constraints is *CPU-time expensive*. Sub-section 6.2 discusses this issue in more detail.

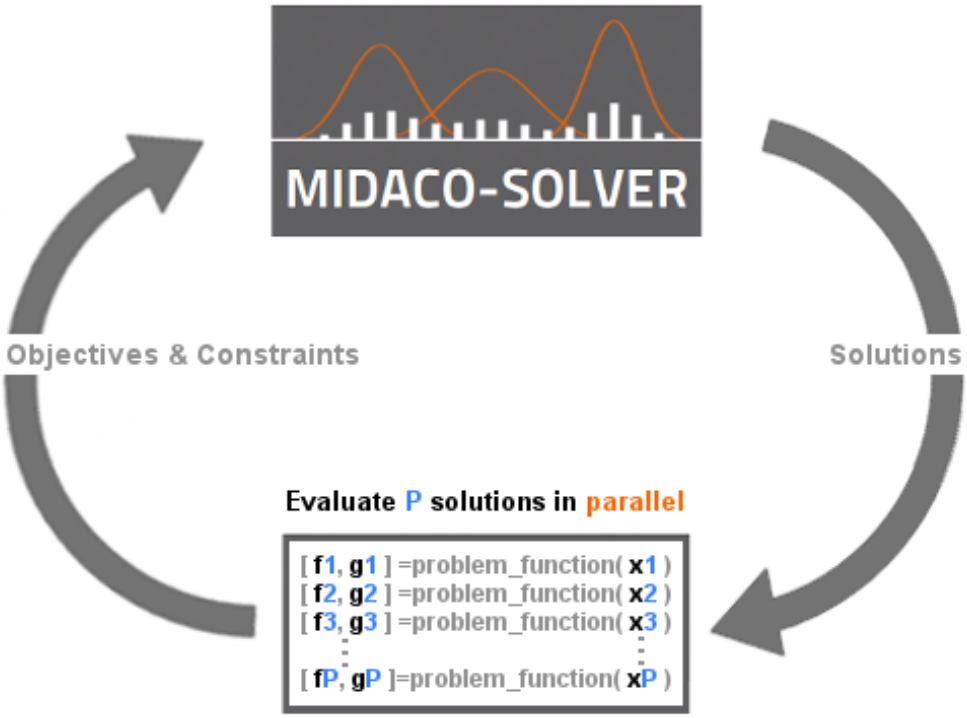


Figure 4: MIDACO evaluating a block of **P** solutions in parallel.

6.1 Running MIDACO in parallel

Running MIDACO in parallel mode is easy. Fully functional examples are given online for various languages (Matlab, Python, C++, R, Java, C#, Fortran) at:

http://www.midaco-solver.com/index.php/more/parallelization

Note that MIDACO's parallelization feature is not limited to those languages and approaches (like openMP or MPI) and can further be used in other languages and other parallelization approaches (like GPGPU or Hadoop/Spark). Based on MIDACO's reverse communication concept its parallelization feature can be enabled with virtually any language/approach.

6.2 Parallelization overhead

Because parallel computing will introduce some computational overhead, the question if running MIDACO in parallel mode is effective or not depends mainly on two aspects: The programming language/approach and the actual CPU-time to evaluate the objectives and constraints. As the computational overhead can be significantly large in some languages (particular in Matlab), it may happen that parallelization actually increases the overall optimization time, instead of reducing it. This is normally the case if problem function evaluation is not time intensive to compute, for example a single evaluation takes less than 0.00001 seconds.

Table 4 gives some rough guideline on the minimal evaluation cost for which parallelization is recommended, depending on various languages. As those times are subject to the actual cpu specification and also the number of available parallel threads, the actual times for a user may be both: larger or smaller. Users are advised to experiment, if parallelization in a given case is beneficial or not.

Table 4: Minimal evaluation cost for which parallelization is promising

Language	Minimial evaluation cost
Matlab	1.0 Second
Python	0.01 Second
R	0.01 Second
Java	0.001 Second
C/C++	0.001 Second
Fortran	0.001 Second

7 Tips & Tricks

This section describes some advanced hints in common optimization scenarios.

7.1 Constraint Handling

Constraint handling is an important and challenging area in optimization. The MIDACO software is capable to solve problems with up to hundreds and even thousands of constraints (e.g. see MIDACO benchmark website or Schlueter and Munetomo [31]). This includes equality as well as in-equality constraints, whereas in-equality constraints are normally easier to solve by MIDACO. Due to MIDACO's black-box concept there is no restriction on the function properties of the constraints, therefore they might be linear or non-linear and do not need to be smooth or differentiable.

In order to efficiently solve problems with many and/or difficult constraints with MIDACO, a *cascading* approach with multiple runs is recommended. The most critical parameter in solving problems with many constraints is the accuracy (PARAM(1), see Section 4.1) that measures the constraint violation. For problems with many constraints the default accuracy of 0.001 might be too difficult or time intensive to reach with a solution from scratch. Therefore the accuracy should be increased for a first run. Depending on the given application, a suitable accuracy value for a first run might be for example 0.1, 0.5 or even 1.0. With such high PARAM(1) value MIDACO will normally find feasible solutions much quicker and will also proceed faster in minimizing the objective function value, once a feasible area is found.

After above described first run with moderate constraint accuracy, the refinement of the solution accuracy can begin. Above implied feasible solution with moderate constraint accuracy can be used as starting point for further runs with a more precise accuracy, such as 0.01, 0.001 or lower. For such refinement runs it is advisable to activate the FOCUS parameter (see Section 4.6). How many refinement runs and which particular ACC and FOCUS settins are promising is subject to a given application. Table 5 gives a rough example on possible ACC and FOCUS settings for solving a difficult constrained problem in several runs. Note that in Table 5 it is assumed that the final solution satisfies a constraint violation equal or below 0.00001.

Table 5: Potential settings for multiple runs

Run	ACC	FOCUS	Starting point
1st	0.5	0.0 (default)	from scratch
2nd	0.1	-10.0	previous solution
3rd	0.01	-100.0	previous solution
4th	0.001	-1000.0	previous solution
5th	0.00001	-100000.0	previous solution

With the hypothetical multiple run setup in Table 5 it requires five runs in total to solve the constrained problem to a solution with the desired constraint accuracy of 0.00001. Note that the FOCUS parameter is used with its "-" flag, which forces MIDACO to stay with the current solution and disable complete internal restarts. This is done as in above scenarios the 2nd till 5th run are considered as refinement runs only.

7.2 Highly nonlinear problems

Similar to solving problems with many and/or difficult constraints, it may be useful to solve highly nonlinear problems (and in generally very difficult to solve problems) with a setup of several runs. In such scenario the first run act as *from scratch* run to deliver a decent first solution which is then further refined in following runs. The critical parameter for such refinement runs is FOCUS (see Section 4.6). Table 6 gives an example scenario of three runs where the 2nd and 3rd run are refinement runs.

Table 6: Potential settings for multiple runs

Run	FOCUS	Starting point
1st	0.0 (default)	from scratch
2nd	100.0	previous solution
3rd	-10000.0	previous solution

Note that in Table 6 the 2nd run assumes the FOCUS parameter without the "-" flag. This is done to enable MIDACO to still explore further areas in the 2nd run, even though it is a refinement run. In contrast to that, the 3rd run assumes FOCUS with its "-" flag, as the 3rd run is intended as the final refinement for high precision.

7.3 Large-Scale Problems

Performance on problems with hundreds and thousands of variables generally benefit from tuning the following parameter: FOCUS, ANTS, KERNEL. For the FOCUS parameter values such as 10, 50 or 100 might work in a first run from scratch. The reason for this is that in large-scale problems the FOCUS parameter effect can be stronger as in small-scale problems because a search space reduction is of greater impact in large-scale problems. For the ANTS and KERNEL parameters, settings such as [ANTS=2,KERNEL=2], [ANTS=5,KERNEL=20] or [ANTS=10,KERNEL=50] might improve the performance. Low values for the ANTS and KERNEL parameters will also reduce the search space exploration and thus lead to faster convergence.

On the contrary, above tuning examples might lead to sub-optimal convergence to a local solution. As large-scale problems are generally difficult to solve, such local solution might however be acceptable in such scenarios where the gained reduction in run-time is of greater value than the solution quality.

7.4 CPU-Time expensive applications

The performance on solving CPU-time expensive applications with MIDACO can be greatly improved by using parallelization. For applications where a single objective and constraint evaluation is expensive (for example takes more than few seconds), parallelization will on average always (drastically) reduce the total time required to solve the problem. Furthermore, the higher the level of parallelization the better. In Schlueter and Munetomo [31] it was demonstrated the for a parallelization factor of 100 the number of sequential steps required by MIDACO could be reduced by around 70 times. For a CPU-time expensive application this means that if a cluster with 100

threads is available, the MIDACO runtime can be increased around 70 times. This is for example less than a day instead of 2 month ($\approx$ 60 days).

In addition to parallelization, the same recommendation (ANTS, KERNEL, FOCUS) as given for large-scale problems (see Section 7.3) can be applied to CPU-time expensive applications. This is because in both scenarios, large-scale and CPU-time expensive applications, a reduction of the search space (and therefore of the number of function evaluation) has great impact. As mentioned in (see Section 7.3) the recommended settings might lead to faster convergence but a sub-optimal solution. For CPU-time expensive application this might be acceptable when a sub-optimal but good solution reached in reasonable time is preferred over a global solution which search effort would require unreasonable time.

7.5 Solving non-linear equation systems

Due to MIDACO's capability to handle problems with hundreds and even thousands of constraints, MIDACO can be used to solve systems of nonlinear equations (in which there are typically no particular objective functions). Instead of formulating all constraints as constraints and leaving the objective function blank (for example a constant value), it is recommended that the most difficult constraint is formulated as objective. Such way MIDACO will more easily find solutions that satisfy the majority of constraints and future refinement runs can focus on satisfying the most difficult constraint (given as objective).

7.6 Joining multiple pareto front files

Joining multiple pareto front files can be useful if several runs on a multi-objective problem are performed and their results should be merged. Joining multiple pareto front files can be achieved by simply appending the solution content from one "MIDACO_PARETOFRONT.TXT" file to another one. The PlotTool will automatically recognize and plot all solutions given in the file, regardless of the declared "psize" value (which is the original number of solutions). However, caution is advised that the format of the pareto file is intact, that means that line 17 and following contain only the raw solution data [F,G,X] and no comments, empty lines or other data.

Note: Solution history files can be joined in the same manner.

7.7 Plotting the history file

After successful creation of a "MIDACO_HISTORY.TXT" file by setting SAVE2FILE=2, the content can be plotted like a "MIDACO_PARETOFRONT.TXT" file via the PlotTool. Simply replace "MIDACO_PARETOFRONT.TXT" as source file by "MIDACO_HISTORY.TXT".

7.8 Plotting variable relationships

It is possible to use the PlotTool (see Section 5) to illustrate not only the relationship between objectives, but also between variables and constraints. This affects multi-objective as well as single-objective problems and requires a little "hack" of the text file containing the solutions to be plotted (e.g. MIDACO_PARETOFRONT.TXT or MIDACO_HISTORY.TXT). MIDACO will automatically write down the number of objectives (denoted as O) in the head of the text file. By manually increasing this number to a larger value (particular new O value = O+M+N) the user has the option to illustrate any of the solution elements.

For example: Consider an unconstrained single-objective problem with 100 variables. The task is to illustrate the relationship between the 99th and 100th variable in respect to the objective function value. If MIDACO is executed with the option SAVE2FILE=2 set it will create the MIDACO_HISTORY.TXT file, which stores all solutions. By manually changing the number O denoting the number of objectives in this file to 101 (instead of 1), the user has the option to treat and thus plot any of the variables just like an objective. If the 99th and 100th variable should be printed, the input in the *X-Axis* of the PlotTool should be 100 and the *Y-Axis* should be 101. This can be easily enterd as text in the PlotTool (instead of using the drop down menu). Both entered values are increased by their index by 1 because the very first value of each solution is occupied by the objective function value. If those those 99th and 100th variable should be plotted against each other in respect to the objective function value, the *colorbar* objective in the PlotTool should be set to "Objective 1". Figure 5 illustrates the input for this example on the X and Y Axis in the PlotTool.

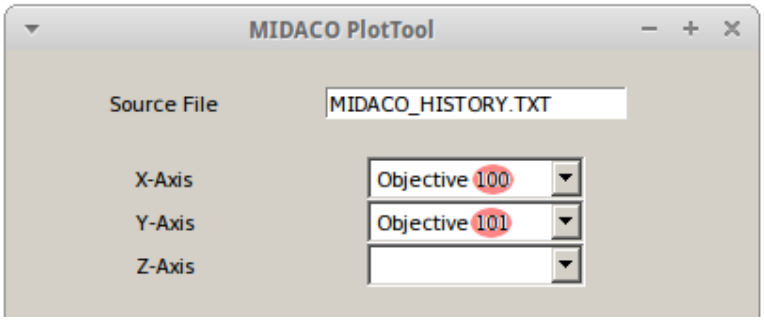


Figure 5: Example input in PlotTool for modified text source file.

Note by applying above described modifications, any direct relationship between variables, constraints and objectives can be illustrated with the PlotTool.

7.9 Multi-modal optimization

Multi-modal optimization seeks to locate not only a single global solution, but the set of all local solutions. Because MIDACO is based on an evolutionary algorithm it explores a vast area of the search space and consequently enters a lot of local optima (in case of multi-modal problem landscapes). Enabling the creation of a history file (see Section 2.2 and Section 7.7) gives the user an easy option to keep track on all evaluated solutions and consequently to further investigate all local solutions found during the search process.

7.10 Submitting several starting points

When running MIDACO with parallelization, it is possible to submit several starting points. This option is useful for very cpu-time intensive application (which require e.g. hours for a single evaluation). By default, MIDACO example templates expect only a single starting point, which is then duplicated into the XXX array which stores P solution vectors X one after another. The corresponding source code in Matlab can be found in the gateway "midaco.m" around line 100:

```
103  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104  xxx = zeros(1,P*n);
105  for c = 1:P
106      for i = 1:n
107          xxx((c-1)*n+i) = x0(i);
108      end
109  end
110  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In case several starting points should be stored in the XXX array, those must be placed manually by the user by modifying the XXX fill up command. For example, consider P=3 and three different starting points, then the modified "midaco.m" gateway might look like following:

```
103  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104  xxx = zeros(1,P*n);
105  % Fill up XXX array with 3 different starting points
106  for i = 1:n
107      xxx( 0*n + i ) = starting_point_1_(i);
108      xxx( 1*n + i ) = starting_point_2_(i);
109      xxx( 2*n + i ) = starting_point_3_(i);
110  end
111  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The source code commands for the XXX array fill up might look slightly different depending on the language, but is essentially always fulfilling the same purpose. Note that it is also possible (and can be useful) to submit different random solutions as starting points.

7.11 Parallel-Overclocking with MIDACO

In case of CPU-time expensive applications where the evaluation time drastically varies (for example some evaluation may take seconds while other may take minutes or hours) and parallelization with a sufficient large number of threads/cores is performed, it can be efficient to *overclock* MIDACO's parallelization factor. The term *overclocking* means here that a larger parallelization factor P is assigned than actual physical threads/cores are available on the CPU-system. Because MIDACO's parallelization factor P can be freely chosen, it can be easily set to any integer value larger (or smaller) than the actual number of threads/cores available.

For example: Consider a cluster of 32 CPU's (each with one core) which act as function evaluator for some master node which runs MIDACO (such setup can for example be established with Spark parallelization in Python). Assigning a parallelization factor of P=64 or P=128 to MIDACO will exceed the actual number of number of available cores but might lead to overall faster processing. The reason is that the pool of cores in the cluster can be used more effectively by utilizing cores which otherwise (in case of P=32) would be idle for some time after a evaluating a fast calculating solution.

Note: In some cases overclocking might also be effective on a single machine. However, overclocking is not recommended on applications where all solution evaluation require equal or quite similar CPU-time.

8 IFLAG Messages

This section describes the list of IFLAG values used by MIDACO as *information flag*. MIDACO reports various IFLAG values to indicate final solution information, warnings or input errors. In case of final solution messages an IFLAG value between 1 and 10 is stated, indicating the reason for terminating and the feasibility of the solution. It is quite common in the process of setting up a new optimization problem that some IFLAG error messages (like IFLAG=204 → bound error) are experienced. Those are normally easy to fix and not of greater concern. The following sub-sections list all IFLAG values.

8.1 Solution Messages (IFLAG = 1 ∼ 9)

Table 7 describes IFLAG messages which are reported along with a solution reported by MIDACO.

Table 7: MIDACO solution messages indicated by IFLAG

IFLAG	
1	Feasible solution found, MIDACO was stopped by MAXEVAL or MAXTIME
2	Infeasible solution found, MIDACO was stopped by MAXEVAL or MAXTIME
3	Feasible solution, MIDACO stopped automatically by ALGOSTOP
4	Infeasible solution, MIDACO stopped automatically by ALGOSTOP
5	Feasible solution, MIDACO stopped automatically by EVALSTOP
6	Infeasible solution, MIDACO stopped automatically by EVALSTOP
7	Feasible solution, MIDACO stopped automatically by FSTOP

8.2 Warning Messages (IFLAG = 10 ∼ 99)

Table 8 describes IFLAG messages which are reported as warning at the beginning of the optimization process. Those warning can be ignored, but are sometimes an indicator that the problem setup is flawed.

Table 8: MIDACO warning messages indicated by IFLAG

IFLAG	
51	Some X(i) is greater/lower than +/- 10^{16} (try to avoid huge values)
52	Some XL(i) is greater/lower than +/- 10^{16} (try to avoid huge values)
53	Some XU(i) is greater/lower than +/- 10^{16} (try to avoid huge values)
71	Some XL(i) = XU(i) (fixed variable)
81	F(1) has value NaN for starting point X
82	Some G(X) has value NaN for starting point X
91	FSTOP is greater/lower than +/- 10^{16}
92	ORACLE is greater/lower than +/- 10^{16}

8.3 Error Messages (IFLAG = 100 ∼ 999)

Table 9 and Table 10 describe IFLAG messages which are reported as error at the beginning of an optimization run. MIDACO will reject to perform any optimization if an error message is raised.

Table 9: MIDACO error messages indicated by IFLAG

IFLAG	Message Description
100	P <= 0 or P > 10^{99}
101	O <= 0 or O > 10^{9}
102	N <= 0 or N > 10^{99}
103	NI < 0
104	NI > N
105	M < 0 or M > 10^{99}
106	ME < 0
107	ME > M
201	Some X(i) has type NaN
202	Some XL(i) has type NaN
203	Some XU(i) has type NaN
204	Some X(i) < XL(i)
205	Some X(i) > XU(i)
206	Some XL(i) > XU(i)
301	PARAM(1) < 0 or PARAM(1) > 10^{99}
302	PARAM(2) < 0 or PARAM(2) > 10^{99}
303	PARAM(3) greater/lower than +/- 10^{99}
304	PARAM(4) < 0 or PARAM(4) > 10^{99}
305	PARAM(5) greater/lower than +/- 10^{99}
306	PARAM(6) not discrete or PARAM(6) > 10^{99}
307	PARAM(7) < 0 or PARAM(7) > 10^{99}
308	PARAM(8) < 0 or PARAM(8) > 100
309	PARAM(7) < PARAM(8)
310	PARAM(7) > 0 but PARAM(8) = 0
311	PARAM(8) > 0 but PARAM(7) = 0
312	PARAM(9) greater/lower than +/- 10^{99}
321	PARAM(10) >= 10^{99}
322	PARAM(10) not discrete
331	PARAM(11) < 0 or > 0.5
344	Pareto front (PF) workspace **LPF too small**. LPF must be at least of size (O+M+N)*PARETOMAX + 1, where PARETOMAX=100 is default (see PARAM(10))
347	PARAM(5) > 0 but PARAM(5) < 1
348	PARAM(5): Optional EVALSTOP precision appendix > 0.5
351	PARAM(12) < 0 or PARAM(12) > 3
399	Some PARAM(i) has type NaN
401	ISTOP < 0 or ISTOP > 1
402	Starting point does not satisfy all-different constraint

Table 10: MIDACO error messages indicated by IFLAG (continued)

IFLAG	Message Description
501	Double precision work space size **LRW is too small**. Increase size of RW array. RW must be at least of size LRW = 105*N+M*P+2*M+O*O+4*O*P+10*O+3*P+610
601	Integer work space size **LIW is too small**. Increase size of IW array. IW must be at least of size LIW = 3*N+P+110
701	Input check failed! MIDACO must be called initially with IFLAG = 0
881	Integer part of X contains continues (non discrete) values
882	Integer part of XL contains continues (non discrete) values
883	Integer part of XU contains continues (non discrete) values
900	Invalid or corrupted LICENSE-KEY
999	N > 4. The free test version is limited up to 4 variables.

Acknowledgement

The first author acknowledges the professional and financial support by the European Space Agency (ESA-ESTEC/Contract No.21943/08/NL/ST) and EADS Astrium Ltd (Stevenage, UK) on the MIDACO development. The first author is furthermore grateful to the Japanese Space Exploration Agency (JAXA) and the Riken Advanced Institute for Computational science (Proposal No.hp140231) for their support on the development and numerical evaluation of the *Utopia-Nadir-Balance* concept for many-objective optimization on the K-Supercomputer facilities.

References

[1] Alghamdi W.Y., Wu H., Zheng W., Kanhere S.S.: *Constructing A Shortest Path Overhearing Tree With Maximum Lifetime In WSNs.* Hawaii International Conference on System Sciences (HICSS-49) (2016)

[2] Allugundu I., Puranik P., Lo Y.P. and Kumar A.: *Acceleration of distance-to-default with hardware-software co-design.* 22nd International Conference on Field Programmable Logic and Applications (FPL) (2012)

[3] Astos Solutions GmbH: *Low-Thrust Orbit Transfer Trajectory Optimization Software (LOTOS)*. Stuttgart, Germany (2016)

[4] Bahbahani M.S., Baidas M.W., Alsusa E.A.: *A Distributed Political Coalition Formation Framework for Multi-Relay Selection in Cooperative Wireless Networks.* IEEE Transactions on Wireless Communications, Volume 14 , Issue: 12, pp. 6869 - 6882 (2016)

[5] Bahbahani M.S., Alsusa E.A.: *Relay Selection for Energy Harvesting Relay Networks using a Repeated Game.* IEEE Wireless Communications and Networking Conference (WCNC), At Doha, Qatar (2016)

[6] Baidas M.W. and MacKenzie A.B.: *On the Impact of Power Allocation on Coalition Formation in Cooperative Wireless Networks.* IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (2012)

[7] Baidas M.W. and Alsusa E.A.: *Power allocation, relay selection and energy cooperation strategies in energy harvesting cooperative wireless networks.* Wirel. Commun. Mob. Comput., DOI: 10.1002/wcm.2668 (2016)

[8] Baidas M.W. and Masud M.: *Energy-efficient partner selection in cooperative wireless networks: a matching-theoretic approach.* International Journal of Communication Systems, Volume 29, Issue 8, pp 1451-1470 (2016)

[9] Chagwiza G., Musekwa S., Jones B., Mtisi S.: *Impact of new water sources on the overall water network: an optimisation approach.* International Journal of Mathematics and Statistics Research, Vol.1, No.1, pp. 32-41 (2014)

[10] Comas M.: *Application of sales forecasting for new products.* Technical report, Escola tecnica superior d'enginyeria industrial de Barcelona, Spain (2012)

[11] Dell I., Lekszyck T., Pawlikowski M., Grygoruk R., Greco L. : *Designing a light fabric metamaterial being highly macroscopically tough under directional extension: rst experimental evidence.* Zeitschrift fur angewandte Mathematik und Physik (ZAMP) Vol 66 (6), pp. 3473-3498 (2015)

[12] Esche E.: *MINLP optimization under uncertainty of a mini plant for the oxidative coupling of methane.* PhD-Thesis, Fakultaet III, Prozesswissenschaften, Technical University of Berlin (2015)

[13] European Space Agency (ESA) and Advanced Concepts Team (ACT): GTOP database - global optimisation trajectory problems and solutions. (2016)

[14] Grujic I., Nilsson R.: Model-based development and evaluation of control for complex multi-domain systems: attitude control for a quadrotor UAV. Technical report ECE-TR-23, Aarhus University, Denmark (2016)

[15] Haenel M., Kuhn S., Henrich D., Gruene L. and Pannek J.: *Optimal camera placement to measure distances regarding static and dynamic obstacles.* Int. J. of Sensor Networks, 12(1), pp.25–36 (2012)

[16] Hunter, J. D.: *Matplotlib: A 2D graphics environment.* Computing In Science & Engineering, 9(3), pp.90–95 (2007)

[17] Panduragan V.: *Optimization and Supervisory Control of Cogeneration Systems.* PhD-Thesis, Dep. Electrical and Computer Engineering, University of Calgary, Canada (2013)

[18] Lou X.: *Acceleration of Distance-to-Default with GPU.* Master-Thesis, School of Information & Communication Technology Royal Institute of Technology Stockholm, Sweden (2012)

[19] Rehberg M., Ritter J.B, Genzela Y., Flockerzi D. and Reichl U.: *The relation between growth phases, cell volume changes and metabolism of adherent cells during cultivation.* J. Biotechnol., 164(4), pp. 489–499 (2013)

[20] Schittkowski K.: *NLPQLP - A Fortran Implementation of a Sequential Quadratic Programming Algorithm with distributed and non-monotone Line Search (User Manual)*, Report, Department of Computer Science, University of Bayreuth (2009)

[21] Schlueter M., Egea J.A. and Banga J.R.: *Extended Ant Colony Optimization for non-convex Mixed Integer Nonlinear Programming*, Comput. Oper. Res. 36(7), pp. 2217–2229 (2009)

[22] Schlueter M., Egea J.A., Antelo L.T., Alonso A.A. and Banga J.R.: *An extended Ant Colony Optimization algorithm for integrated Process and Control System Design*, Ind. Eng. Chem. 48(14), pp. 6723–6738 (2009)

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

[24] Schlueter M., Rueckmann J.J and Gerdts M.: *A Numerical Study of MIDACO on 100 MINLP Benchmarks*, Optimization, 61(7), pp. 873–900 (2012)

[25] Schlueter M.: *Nonlinear mixed integer based Optimisation Technique for Space Applications*, Ph.D. Thesis, School of Mathematics, University of Birmingham (UK) (2012)

[26] Schlueter M., Erb S., Gerdts M., Kemble S. and Rueckmann J.J.: *MIDACO on MINLP Space Applications*, Optimization, 51(7), pp.1116–1131 (2013)

[27] Schlueter M. and Munetomo M.: *Parallelization Strategies for Evolutionary Algorithms for MINLP*, Proc. Congress on Evolutionary Computation (IEEE-CEC), pp.635-641 (2013)

[28] Schlueter M. and Munetomo M.: *Parallelization for Space Trajectory Optimization*, Proc. World Congress on Computational Intelligence (IEEE-WCCI), pp. 832 - 839 (2014)

[29] Schlueter M.: *MIDACO Software Performance on Interplanetary Trajectory Benchmarks*, Advances in Space Research (Elsevier), Vol 54, Issue 4, Pages 744 - 754 (2014)

[30] Schlueter M., Yam C.H., Watanabe T., Oyama A.: *Parallelization Impact on Many-Objective Optimization for Space Trajectory Design*, *International Journal of Machine Learning and Computing 6.1: 9-14.* (2016)

[31] Schlueter M. and Munetomo M.: *Numerical Assessment of the Parallelization Scalability on 200 MINLP Benchmarks*, *Proc. IEEE-WCCI, Vancouver, Canada* (2016)

[32] Takano A.T. and Marchand B.G.: *Optimal Constellation Design for Space Based Situational Awareness Applications* AAS/AIAA Astrodynamics Specialists Conference (Paper No. AAS11-543) (2011)

[33] Ukritchon B., Boonyatee T.: *Soil Parameter Optimization of the NGI-ADP Constitutive Model for Bangkok Soft Clay.* Geotechnical Engineering Journal of the SEAGS & AGSSEA, Vol. 46 No.1, pp. 28-36 (2015)

[34] Weiss L., Koeke H., Schlueter M., Huehne C.: *Structural optimisation of a composite aircraft frame for a characteristic response curve*. Proc. European Conference on Composite Materials (ECCM17), Munich, Germany (2016)

[35] Wong S.I.: *On Lightweight Design of Submarine Pressure Hulls.* MSc Thesis, Delft University of Technology (2012)