saCeSS optimization library - Documentation for version 0.1 - R2016A

David R. Penas*[1], Patricia González†[2], Jose A. Egea‡[3], Ramón Doallo§[2], and Julio R. Banga¶[1]

[1]BioProcess Engineering Group, IIM-CSIC, Vigo (Spain)
[2]Computer Architecture Group, Universidade da Coruña (Spain)
[3]Department of Applied Mathematics and Statistics, Universidad Politécnica de Cartagena (Spain)

October 7, 2016
Version 17

*david.penas@iim.csic.es
†patricia.gonzalez@udc.es
‡josea.egea@upct.es
§doallo@udc.es
¶julio@iim.csic.es

Contents

1 Abstract

Here we present the documentation for a novel parallel global optimization code, self-adaptive cooperative enhanced scatter search (saCeSS). This code is distributed as a library with several parallel solvers based on the scatter search metaheuristic, incorporating several key new mechanisms: (i) asynchronous cooperation between parallel processes, (ii) coarse and fine-grained parallelism, and (iii) self-tuning strategies. The saCeSS code has been implemented using Fortran 90 and C. Parallelization has been implemented using MPI and openMP. It has been tested in Linux clusters running CentOS 6.7 and Debian 8.

The saCeSS library allows the solution of non-linear programming (NLP) and mixed-integer non-linear programming (MINLP) problems. It also provides efficient local solvers for nonlinear parameter estimation problems associated with complex models (e.g. those described by differential equations). The current distribution of saCeSS includes a set of optimization examples that can be used as benchmarks, taken from the BBOB and BioPreDyn testbeds.

2 Introduction

Many questions of interest in science and engineering can be formulated as global optimization problems. Although stochastic global optimization methods can be used to solve many of such problems in reasonable computation times, very often we must face situations where the associated computational cost is very significant. Moreover, many of these methods need the tuning of a number of adjustable search parameters, requiring a number of initial exploratory runs and therefore further increasing the computation times. Fortunately, parallelization strategies can help us to surmount these difficulties.

Enhanced scatter search (eSS) is a promising metaheuristic that in sequential implementations has been shown to outperform other state of the art stochastic global optimization methods, especially in parameter estimation problems [1–8]. Recently, a prototype of cooperative scatter search (CeSS) implementation using multiple processors was presented [9], showing good performance for the calibration of several large-scale models. However, this prototype used a simple synchronous strategy and small number of processors (due to inefficient communications). Thus, although it could reduce the computation times of sequential scatter search, it still required very significant efforts when dealing with large-scale applications.

Here we present a new parallel global optimization library named self-adaptive cooperative enhanced scatter search (saCeSS). This code is distributed as a library with different parallel solvers based on the scatter search metaheuristic, incorporating several key new mechanisms: (i) asynchronous cooperation between parallel processes, (ii) coarse and fine-grained parallelism, and (iii) self-tuning strategies. The saCeSS code has been implemented using Fortran 90 and C. Parallelization has been implemented using MPI and openMP. It has been tested in Linux clusters running CentOS 6.7 and Debian 8.

The saCeSS library allows the solution of non-linear programming (NLP) and mixed-integer non-linear programming (MINLP) problems. It also provides efficient local solvers for nonlinear parameter estimation problems associated with complex models (e.g. those described by differential equations). The excellent performance and scalability of this novel method are illustrated considering a set of very challenging benchmark problems. The results consistently show that saCeSS is a robust and efficient method, allowing very significant reduction of computation times with respect to previous methods (from days to minutes, in several cases) even when only a small number of processors is used.

3 Problem statement

The saCeSS optimization library contains global optimization methods based on enhanced scatter search (eSS) [1–8], which seek the global minimum of non-linear programming (NLP) and even mixed-integer nonlinear programming (MINLP) problems specified by

$$\min_x f(x, p_1, p_2, ..., p_n) \tag{1}$$

subject to

$$c_{eq} = 0 \tag{2}$$

$$c_L \leq c(x) \leq c_U \tag{3}$$

$$x_L \leq x \leq x_U \tag{4}$$

where x is the vector of decision variables, and x_L and x_U its respective bounds. p_1,...,p_n are optional extra input parameters to be passed to the objective function. c_{eq} is a set of equality constraints. $c(x)$ is a set of inequality constraints with lower and upper bounds, c_L and c_U. Finally, $f(x, p_1, p_2, ..., p_n)$ is the objective function to be minimized.

In order to solve an optimization problem with a solver from the saCeSS optimization library, the following information must be provided:

- **f**: Name of the objective function.
- **dim**: Number of decision variables.
- **x_L**: Vector containing the lower bounds of the variables.
- **x_U**: Vector containing the upper bounds of the variables.
- **x_0**: Vector containing the given initial point (optional).
- **vtr**: Objective function value to be reached (optional).

If the problem contains additional contraints and/or integer or binary variables, the following information should also be defined:

- **neq**: Number of equality (=0) constraints.
- **c_L**: Vector defining the lower bounds of the inequality constraints.
- **c_U**: Vector defining the upper bounds of the inequality constraints.
- **int_var**: Number of integer variables.
- **bin_var**: Number of binary variables.

The stopping condition of saCeSS is based upon three different criteria: maximum computation time, maximum number of function evaluations and a value to reach (see Section 8.1). The third one (value to reach) is problem dependent and not mandatory if the user has no prior knowledge about the optimal values of the incumbent problem. saCeSS will stop when any of these three criteria is fulfilled. More information about stopping criteria in the framework of scatter search can be found in the eSS user manual (`http://gingproc.iim.csic.es/SSm-eSS_userguide.pdf`)

A more detailed description of the input options and formatting is given in section 8.

4 Features and structure

The saCeSS library has been implemented using Fortran 90 and C, with parallel programming using both MPI and openMP directives.

The library includes a set of different versions of Scatter Search can be used to applied to global optimization problems:

- eSS: sequential version of enhanced Scatter Search [2]. Most input options for eSS were implemented in this version.
- eSSm: eSS with multiple configurations, where a specific number of instances of sequential scatter search are executed in parallel without cooperation among them. This can be useful in order to evaluate the parallel cooperative versions below.
- CeSS: cooperative enhanced Scatter Search, i.e. a parallel cooperative eSS scheme where different eSS processes run in parallel and exchange information in a synchronous fashion (based on a pre-defined time intervals), as described by Villaverde et al. [9].
- aCeSS: asynchronous Cooperative enhanced Scatter Search, a distributed asynchronous version, as described in [10].
- Self-adaptive asynchronous Cooperative enhanced Scatter Search (saCeSS). This is the most recent, and more competitive, version, which includes mechanisms of self-adaptation. **In principle this should be the version performing the best (in terms of efficiency and robustness), so it is the one recommended for most users**.

The execution of all these solvers produces the following output:

- a main output file, reporting general information, e.g. optimization problem characteristics, search options enabled, final solution achieved, final computation time, etc.
- a more specific log (per processor) which can be useful to analyse the distributed behaviour of the algorithm.
- output plots: convergence plots and, in the case of saCeSS algorithm, Gantt charts illustrating cooperation among processors, and plots detailing the search improvements history.

The code is organized in several folders as follows:

- benchmark: contains the source code of the different optimization problems integrated in the solver. If you want to add a new problem, you should modify the template files in the folder *customized*.
- doc: documentation generated by Doxygen, in several formats (including HTML).
- include: header files of C functions.
- src: Fortran 90 and C files, with I/O modules.
- inputs: set of templates and examples for the required XML input file.
- lib: set of required libraries, such as BLAS, hdf5, libxml2, GSL, etc.
- output: folder for the results files generated by the executions.

The diagram in Figure 1 summarizes the general architecture of the saCeSS optimization library.

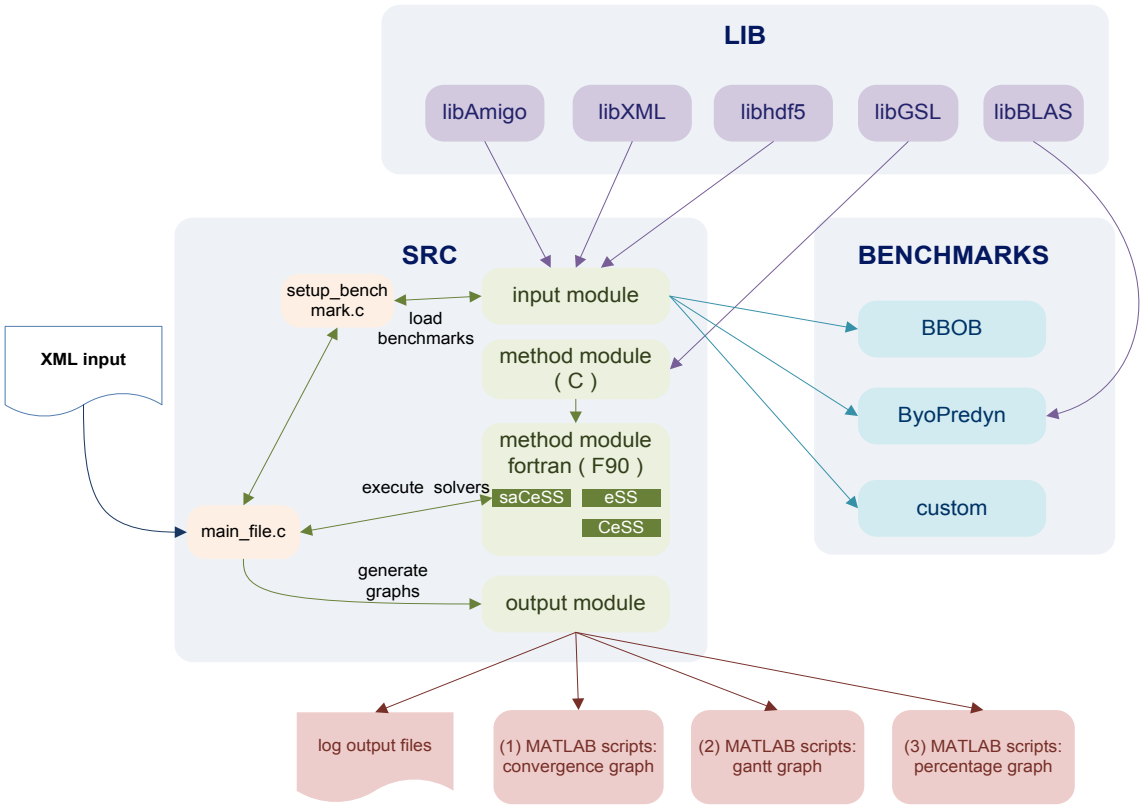

Figure 1: Architecture of saCeSS optimization library.

4.1 Local solvers

Local solvers are a key element in Scatter Search, greatly accelerating convergence. Although Scatter Search can, in principle, be executed without a local solver, this is only recommended for extremely pathological cost functions (e.g. very noisy, or with many sharp discontinuities). The local solvers provided with the current version of saCeSS, are:

- *nl2sol*, which stands for "Nonlinear Least-Squares Algorithm"[11]. This solver uses the Jacobian of the residual vector to approximate and iteratively improve the input solution. The version used is the nl2sol solver from the PORT library.
- *DHC*, which stands for "Dynamic Hill Climbing"[12]. It is a direct search method recommended for those problems in which the objectived and/or constraint function are very noisy, or when the gradient information is difficult to approximate accurately. It can be regarded as a more robust choice for those problems where nl2sol fails or performs badly.
- *MISQP*, which stands for "Mixed Integer Sequential Quadratic Programming"[13]. This is the local method recommended for MINLP problems.

5 Requirements

The current version of saCeSS has the following requirements:

- Linux command line systems (saCeSS has been tested under CentOS 6.6, CentOS 6.7 and Debian 8).
- GNU or Intel compilers. saCeSS has been compiled with gcc-4.4.7, but it has been also tested with Intel compilers (icc-13.1.1 and intel-14.0.2).
- openMPI or IntelMPI implementation of message passing interface (MPI). saCeSS has been compiled with openmpi-1.8.3 and openmpi-1.8.4. It has also been tested with intelmpi-5.0.

By default, saCeSS uses a general version of BLAS. However, you can use other implementations of this library, such as that present in Intel MKL. The multi-Processing technology openMP is included by default in gcc and icc compilers.

6 Installation

To install the saCeSS optimization library, please perform the following steps:

(1) untar the code in a directory.

(2) modify the Makefile to set the compilers and the MPI implementations that you want to use. There are five possible combinations between compilers and MPI implementation:

- GNU compilers: to set only the sequential version with GNU compilers, please uncomment the following section, and comment the rest of the compilation sections:

```
########################################################################
## SEQUENTIAL GCC/GFORTRAN
########################################################################
CC:= gcc
FC:= gfortran
CLIBS+= -lstdc++ -lpthread -lrt -lgfortran  -cpp -MMD -lm
CFLAGS+= -O3 -cpp -DGNU
FLIBS+=
FFLAGS+= -O3 -cpp -DGNU
LIBS+= -L$(LIBRARY)/BLAS -lblas
########################################################################
```

- Intel compiler: to set only the sequential version with Intel compilers, please uncomment the following section, and comment the rest of the compilation sections.

```
########################################################################
## SEQUENTIAL ICC/IFORT
########################################################################
CC:=   icc
FC:=   ifort
AR:=   xiar#
LD:=   xild#
CLIBS+=  -cxxlib -lrt -lhdf5  -lxml2 -limf -lifcore  -lm -MMD
CFLAGS+= -O3 -ipo -xHost -DINTEL
FLIBS+=  -cxxlib
FFLAGS+= -O3 -ipo -xHost -fpp -DEXPORT -DINTEL
BLAS+= $(LIBRARY)/BLAS/libblas.a
########################################################################
```

- GNU compilers and openMPI: please uncomment the following section of the makefile, and comment the rest of the compilation sections:

```
########################################################################
## PARALLEL GCC/GFORTRAN - OPENMP
########################################################################
CC:=mpicc
FC:=mpif90
```

```
CLIBS+=  -lstdc++ -lpthread -lrt -lgfortran  -cpp -MMD -lm
CFLAGS+= -O3 -cpp -DOPENMP -DMPI2 -DGNU -fopenmp -lmpi
FLIBS+=
FFLAGS+= -O3 -cpp -DOPENMP -DMPI2 -DGNU -fopenmp
LIBS+= -L$(LIBRARY)/BLAS -lblas
#########################################################################
```

- Intel compilers + IntelMPI: please uncomment the following section, and comment the rest of the compilation sections:

```
#########################################################################
## PARALLEL ICC/IFORT - INTELMPI
#########################################################################
CC:=   mpiicc
FC:=   mpiifort
AR:=   xiar#
LD:=   xild#
CLIBS+=  -cxxlib -lrt -lhdf5  -lxml2 -limf -lifcore
CLIBS+=  -lm -MMD
CFLAGS+= -O3 -ipo -xHost -DOPENMP -DMPI2 -DINTEL
CFLAGS+= -openmp -pthread -mt_mpi
FLIBS+=  -cxxlib
FFLAGS+= -O3 -ipo -xHost -fpp -DEXPORT -mt_mpi
FFLAGS+= -openmp -DOPENMP -DMPI2 -DINTEL -mt_mpi
BLAS+= $(LIBRARY)/BLAS/libblas.a
#########################################################################
```

- Intel compilers + openMPI: please uncomment the following section, and comment the rest of the compilation sections:

```
#########################################################################
## PARALLEL ICC/IFORT - OPENMPI
#########################################################################
OPENMPI:=PATH OPENMPI (compiled for icc/ifort)
CC:=   $(OPENMPI)/bin/mpicc
FC:=   $(OPENMPI)/bin/mpif90
AR:=   xiar#
LD:=   xild#
CLIBS+=  -cxxlib -lrt -lhdf5  -lxml2 -limf -lifcore
CLIBS+=  -lm -MMD
CFLAGS+= -O3 -ipo -xHost -DOPENMP -DMPI2 -DINTEL
CFLAGS+= -openmp -pthread -lmpi
FLIBS+=  -cxxlib
FFLAGS+= -O3 -ipo -xHost -fpp -DEXPORT -lmpi
FFLAGS+= -openmp -DOPENMP -DMPI2 -DINTEL
BLAS+= $(LIBRARY)/BLAS/libblas.a
#########################################################################
```

(3) Check if the all compiler commands and libraries are set in the PATH. Otherwise, the compilation of the library will result in errors. For instance, you might need to export the Intel compilers and libraries (gcc is usually in the PATH by default).

(4) Clean all object files of previous installations by doing:

```
$  make veryclean
```

(5) Create the executable by doing:

```
$  make all
```

(6) Check if the executable named *paralleltestbed* has been created.

```
$  [SACESSROOT]/bin
```

7 Testing the installation

Users can easily test the solvers using the two suites of benchmarks provided with the library:

- BBOB: set of problems from the Black-Box Optimization Benchmarking (BBOB) suite [14].
- ByoPreDyn-bench: set of computational systems biology benchmark problems presented in [15].

There are three basic options to run the solvers, depending whether MPI and openMP are used or not.

(i) To execute the solvers without MPI, use:

```
$  bin/[executable] input/[config_file.xml] output/[output_name]
```

(ii) To execute the solvers with MPI, use:

```
$  export OMP_NUM_THREADS=1
$   -np [number of processors] bin/[executable] input/[config_file.xml]
 output/[output_name]
```

(iii) To execute the solvers with both MPI and openMP, use:

```
$  export OMP_NUM_THREADS=[NUM THREADS]
$  mpirun -x OMP_NUM_THREADS=[NUM THREADS] --map-by slot:pe=[NUM THREADS]
-np [number of processors] bin/[executable] input/[config_file.xml]
  output/[output_name]
```

For the execution of the solvers, users need to indicate an XML input file and the output path where the results logs and plots are going to be saved.

To test the correct installation of saCeSS, we run one of the example problems wiith the following commands:

```
$  export OMP_NUM_THREADS=1
$  mpirun -np 6 bin/paralleltestbed inputs/TEMPLATE_CIRCADIAN.xml output/TEST
```

An example of the typical output for this test problem is shown below. In addition to this output text, in the output path several results files will be created: one Matlab script to plot the convergence curves, a text log file with details of each parallel processor during the execution, and a csv file for each slave with convergence information. In the case of the saCeSS solver, several additional files will be created: a Matlab script to plot a Gantt chart indicating the activity of the processors during the cooperative execution of the run; a csv file with the information used in the Gantt chart; another Matlab script to plot a bar chart with the evolution of the percentage of improvement of the solutions received in the master; and a csv file with the data corresponding to that plot.

```
=====================================================
== PARALLEL SACESS OPTIMIZATION LIBRARY =============
=====================================================

************ GENERAL INFORMATION ********************

* SOLVER :: ScatterSearch
```

```
Version solver : saCeSS - SELF-ADAPTED ASYNCHONOUS COOPERATIVE eSS
Topology       : mixture between master-slave and ring topology
Max.Evals      : 10000000000.0
Max.Time(s)    : 10000000000.0
Local Solver   : nl2sol

* TOTAL PROCESSORS USED :: 6
Number of MPI proc.                : 6
Number of openMP threads per proc. : 1

* TYPE OF BENCKMARK :: systemBiology
Name                      : Circadian problem
Number of parameters      : 13
Value To Reach            : 0.0000100000
[DEFAULT] Value To Reach : 0.0000100000

************ SLAVE INFORMATION ********************
ID SLAVE:   1        Refset size:   5      ndiverse:  130
ID SLAVE:   2        Refset size:   7      ndiverse:  130
ID SLAVE:   5        Refset size:  15      ndiverse:  130
ID SLAVE:   3        Refset size:   9      ndiverse:  130
ID SLAVE:   4        Refset size:  12      ndiverse:  130

************ OPTIMIZATION BEGINS ******************

[PROCESSOR ID=3] Initial Pop: NFunEvals: 135 Bestf: 0.2748440296D+02 CPUTime: 0.27
[PROCESSOR ID=4] Initial Pop: NFunEvals: 135 Bestf: 0.2887226232D+02 CPUTime: 0.28
[PROCESSOR ID=1] Initial Pop: NFunEvals: 135 Bestf: 0.2125707236D+02 CPUTime: 0.28
[PROCESSOR ID=5] Initial Pop: NFunEvals: 135 Bestf: 0.2143410411D+02 CPUTime: 0.28
[PROCESSOR ID=2] Initial Pop: NFunEvals: 135 Bestf: 0.1660158308D+02 CPUTime: 0.28
[MASTER] Best Known Solution fx = 0.0514597656 -- CURRENT TIME 4.501880
[SLAVE ID=2] fx = 0.0000000553 < VTR =0.0000100000 -- CURRENT TIME 4.703395 s
[SLAVE ID=4] fx = 0.0000000556 < VTR =0.0000100000 -- CURRENT TIME 4.995415 s
[SLAVE ID=5] fx = 0.0000000553 < VTR =0.0000100000 -- CURRENT TIME 5.162111 s
[SLAVE ID=3] fx = 0.0000000554 < VTR =0.0000100000 -- CURRENT TIME 6.418595 s

************ RESULTS ****************************

[PROGRAM FINISHED] Desired function value achieved.

ID   0  -- SEED RANDOM NUMBERS:         19421002265000.0
ID   1  -- SEED RANDOM NUMBERS:         38844929600000.0
ID   3  -- SEED RANDOM NUMBERS:         77701559480000.0
ID   4  -- SEED RANDOM NUMBERS:         97134262025000.0
ID   2  -- SEED RANDOM NUMBERS:         58271782005000.0
ID   5  -- SEED RANDOM NUMBERS:        116569889640000.0

BEST SOLUTION:
7.5013044068244374429355048050 64
0.6801268176484495109335171036 95
```

```
10.097054959051927625068856286816
2.342150830257768934217210698989
1.897939860465343464568377385149
1.201052390130370906007328812848
3.805760967695956509260213351808
2.535377841389668596150386292720
0.500431673381125041721873003553
0.002606656011352262609925833914
0.000000000000000000000000000000
0.000000000000000000000000000000
0.000625332716897003494313511673
f(x)=0.00000005530645151681910546431 2

TOTAL TIME TO REACH VTR : 4.703395
TOTAL EVALUATIONS       : 9492
AVERAGE ITERATIONS      : 1.200000
EVALUATION FAILED       : 0

f-0 DIM 13 fbest: 0.00000005530645151682 < ftarget 0.000010000000000000000
numevals (9492) -- time VTR -> 4.703395 -- iter 1.200000

************ GRAPHS GENERATED **********************

GRAPH PATH               : output/TEST
CONVERGENCE GRAPH MATLAB : output/TEST/convergence.m
CONVERGENCE GRAPHs CSV   : output/TEST/convergence_id*.csv
GANTT CHART MATLAB       : output/TEST/gantt_id0.m
GANTT CHARTs CSV         : output/TEST/gantt_id*.csv
PERCENTAGE GRAPH MATLAB  : output/TEST/percentage_id0.m
PERCENTAGE GRAPH CSV     : output/TEST/percentage_id0.csv
```

8 Input options

The following XML code represents a typical template of the required input file. There are three main sections: (1) run metadata, where the general options for the solver and for the benchmark or the user problem are selected, (2) method metadata, which indicates the specific options of the solver and (3) parallelization metadata, which specifies parameters about cooperation.

```
<xml>
  <run>
     <typebench> systemBiology </typebench>
     <id> 0 </id>
     <log_scale> 1 </log_scale>
     <output>  1 </output>
     <verbose> 1 </verbose>
     <local_search> 1 </local_search>
     <stopping_criteria>
        <maxevaluation> 1e10 </maxevaluation>
        <maxtime> 1e10 </maxtime>
        <vtr> 1e-5 </vtr>
     </stopping_criteria>
  </run>
  <method name="saCeSS">
     <user_options>
        <weight> default </weight>
        <tolc> default </tolc>
        <prob_bound>  default </prob_bound>
        <nstuck_solution> default  </nstuck_solution>
     </user_options>
     <global_options>
        <dim_ref> default </dim_ref>
        <ndiverse> default </ndiverse>
        <combination> default </combination>
        <n_stuck> default </n_stuck>
     </global_options>
     <local_options>
        <solver> nl2sol </solver>
        <tol> 2 </tol>
        <evalmax> 10000 </evalmax>
        <iterprint> 0 </iterprint>
        <n1> 1 </n1>
        <n2> 10 </n2>
        <balance> 0.25 </balance>
        <bestx> default </bestx>
     </local_options>
  </method>
  <parallelization name="cooperative">
     <reception_threshold> 1 </reception_threshold>
     <evals_threshold> 5000 </evals_threshold>
     <mult_num_sendSol> 10 </mult_num_sendSol>
     <minimum_num_sendSol> 20 </minimum_num_sendSol>
     <migration_max_time> 10 </migration_max_time>
```

```
  </parallelization>

  <problem>
     <dim> 13 </dim>
     <neq> 0 </neq>
     <ineq> 0 </ineq>
     <int_var> 0 </int_var>
     <bin_var> 0 </bin_var>
     <ub>
       609.79399999999998,71.014399999999995,892.73500000000001,
       238.50400000000002,211.02400000000000,129.91200000000001,
       432.07899999999995,245.27700000000002,49.307600000000001,
       1.0000000000000000,1.0000000000000000,1.0000000000000000,
       1.000000000000000
     </ub>
     <lb>
       6.0979400000000003E-002,7.1014400000000005E-003,8.9273500000000006E-002,
       2.3850400000000001E-002,2.1102400000000000E-002,1.2991200000000001E-002,
       4.3207900000000000E-002,2.4527700000000003E-002,4.9307600000000002E-003,
       0.0000000000000000,0.0000000000000000,0.0000000000000000,
       0.0000000000000000
     </lb>
     <point fx="17.115314239532431">
       1.7390570494392608,1.7487276277823121,0.27129839743060158,
       229.37838065550457,191.68165026936188,12.303964898637144,
       35.379607522118043,95.459437714737945,0.12596775438646635,
       2.0776405322917276E-004,6.4379384917837145E-006,1.5952347084273732E-003,
       5.8297822534562208E-006
     </point>
     <point fx="19.457237536272839">
       2.5515475050419636,18.585351325854564,7.6406367913812927,
       220.10090372386526,19.538101326402657,10.837353478612977,
       191.10600316641649,3.1694329461785435,1.6237942267145693,
       0.62717579957130609,1.4009918251683111E-005,1.2424380744094903E-003,
       1.2452382859735329E-002
     </point>
  </problem>

</xml>
```

In the following subsections we describe each one of the fields in the input XML file.

8.1 Run metadata

The run metadata describes general options for the execution of the solver, such as the selection of the benchmark or user-defined problem, the termination criteria, or the use of local solvers. A detailed description is given in Table 1.

Table 1: **Different fields of run metadata.**

field name	description	values	default
typebench	This field indicates the type of problem selected. There are four possibilities: customized, systemBiology, noiselessBBOB, noiseBBOB.	string	-
id	identification number of the problem. Range depends of the kind of problem selected. Not applicable in customize case	+integer	-
log_scale	enable/disable the search in logarithmic space.	boolean	0
output	enable/disable the output file generation.	boolean	0
verbose	enable/disable the screen messages.	boolean	1
local_search	enable/disable the use of the local solver.	boolean	0
stopping_criteria.maxtime	termination criterion: maximum execution time.	+double (seconds)	-
stopping_criteria.maxevaluation	termination criterion: maximum number of	+integer	-
stopping_criteria.vtr	termination criterion: value to reach (VTR). evaluations.	−/+double	-

Run metadata for benchmark problems: When typebench is assigned the value "systemBiology" (the suite of computational systems biology benchmarks), the id field has a range 0-9, corresponding to following problems:

- ID0 - Circadian model: parameter estimation in a dynamic model of the circadian clock in the plant Arabidopsis thaliana, as presented in [16]. The model consists of 7 ordinary differential equations with 27 parameters (13 of them were estimated) with data sets from 2 experiments.
- ID1 - 3-step pathway model: problem considering a 3-step generic and highly non-linear pathway with 8 differential equations and 36 parameters, and data sets from 16 experiments, as presented in [17].
- ID2 - Nfkb model: this problem is based on the model in [18] and consists of 15 ordinary differential equations with 29 parameters and data sets from 2 experiments.
- ID3 - BioPreDyn B1 problem: calibration of a genome-wide kinetic model of *S. cerevisiae*. It contains 276 dynamic states, 44 observed states and 1759 parameters.
- ID4 - BioPreDyn B2 problem: calibration of a dynamic model of the central carbon metabolism of *E. coli*. It consists of 18 dynamic states, 9 observed states and 116 estimable parameters.
- ID5 - BioPreDyn B3 problem: calibration of a dynamic model of enzymatic and transcriptional regulation of the central carbon metabolism of *E. coli*. It contains 47 dynamic states (fully observed) and 178 parameters to be estimated.

- ID6 - BioPreDyn B4 problem: calibration of a kinetic metabolic model of Chinese Hamster Ovary (CHO) cells, with 34 dynamic states, 13 observed states and 117 parameters.
- ID7 - BioPreDyn B5 problem: calibration of a signal transduction logic model, with 26 dynamic states, 6 observed states and 86 parameters.
- ID8 - BioPreDyn B6 problem: calibration of a dynamic model describing the gap gene regulatory network of the vinegar fly, *Drosophila melanogaster*. It consists of three processes formalized with 108-212 ODEs, and resulting in a model having 37 unknown parameters.

When typebench is assigned the value "noiselessBBOB", the range of ID field is 0-24. For the case of "noiseBBOB", the range is 101-130. Check [14] to see details for each problem ID.

8.2 Method metadata

Table 2 summarizes the method metadata. When one field is marked with ([1]), it means that the string value "default" is used to put by default this options. The current version of saCeSS includes implementations of the following variants of Scatter Search:

- <method name="Scatter Search">: sequential version of eSS.
- <method name="eSSm">: eSS with multiple configuration, where a specific number of instances of sequential scatter search are performed in parallel without cooperation among them.
- <method name="CeSS">: Cooperative enhanced Scatter Search, a parallel cooperative eSS scheme where different sequential eSS instances exchange information synchronously.
- <method name="aCeSS_dist">: Asynchronous Cooperative Scatter Search, a distributed and asynchronous version of eSS.
- <method name="saCeSS">: Self-adapted asynchronous Cooperative Scatter Search, the novel, more competitive, parallel self-adaptive eSS scheme.

Further, it is important to remark that the local options of method metadata are used to select the local solver and its tuning parameters. For the case of continuous global optimization (no integer or binary decision variables), the current version of saCeSS provides two local solvers: DHC ("Dynamic Hill Climbing") and nl2sol, which is a nonlinear least-squares algorithm" (defined in section 4.1).

DHC is a pattern-search (derivative-free) solver easy to apply to all kind of problems, including noisy and large-scale ones, without additional requirements to take into account.

However, nl2sol (which is especially indicated for least squares parameter estimation problems) the cost function declaration is slightly different: it must be declared returning two variables, namely (i) a scalar cost computed as the sum of squares of differences between the experimental and predicted data (i.e., $[\sum_{i=1}^{ndata}(yexp_i - yteor_i)^2]$), and (ii) a vector of those differences, i.e. the residuals (i.e., $R = [(yexp_1 - yteor_1), (yexp_2 - yteor_2), ..., (yexp_{ndata} - yteor_{ndata})]$).

In the current version of the saCeSS library, the residuals for parameter estimation in dynamic models (based on ODEs) have been implemented making use of the inner structures included in the AMIGO toolbox (check the use case describing how to solve a problem declared in AMIGO). However, to apply nl2sol local solver to problems not created using the AMIGO toolbox, the user needs to provide a vector R containing the residuals. We provide an example describing how to perform this in Section 11.7.

Table 2: **Different fields of method metadata.**

field name	description	values	default
method name	This field indicates the version of the solver: ScatterSearch, eSSm, CeSS, saCeSS or aCeSS_dist.	string	-
user options:	options defined by the user.		
[1] weight	Weight for the penalty term added to the objective function in constrained problems.	+double	1e6
[1] tolc	Maximum absolute violation of the constraints.	+double	1e-5
[1] prob_bound	Probability (0-1) of biasing the search towards the bounds.	+double	0.5
[1] nstuck_solution	Number of consecutive iterations without significant improvement before the search stops.	+integer	20
global options:	different options for the global search.		
[1] dim_ref	Number of elements in Refset.	+integer	auto
[1] ndiverse	Number of solutions generated by the diversificator	+integer	10*nvars
[1] combination	Type of combination of Refset elements: (1) hyper-rectangles (2) linear combinations.	+integer +integer	1
[1] n_stuck	Maximum number of iterations without improvement before generating another solution.	+integer	0
local options:	different options for the local search (LS).		
solver	Current options for LS: nl2sol, misqp or dhc. There are special requisites to apply nl2sol to problems not created through AMIGO toolbox.	string	dhc
tol	Level of tolerance in the LS. (1-3)	+integer	2
evalmax	Maximum number of evaluations performed by the LS.	+integer	-
iterprint	Enable/disable the output screen of the LS.	boolean	0
n1	Number of iterations to enter for the first time in the LS.	+integer	1
n2	Number of iterations to enter in LS.	+integer	10
balance	Balances between quality (=0) and diversity (=1) for choosing initial points for the LS.	+double	0.5
finish	Optional parameter. Set a local solver if you want a final refinement with the local solver.	string	0
[1] bestx	When activated (1) local search is only applied to the best solution found so far, ignoring other options.	+integer	0

8.3 Parallelization metadata

These options are only used in the cooperative parallel versions of eSS. The fields with ([2]) are only used with the saCeSS solver. The option with ([3]) only works with the CeSS solver. All the parallelization metadata is ignored in the sequential case of the scatter search algorithm.

Table 3: **Different fields of parallelization metadata.**

field name	description	values	default
[2] reception_threshold	Option to disable/enable the mechanism of automatic management of the reception threshold.	boolean	1
[2] evals_threshold	Parameter to determine whether to reconfigure a slave based on the number of evaluations performed without significantly improving the cooperative solution.	+integer	5000
[2] mult_num_sendSol	Parameter to determine whether to start the reconfiguration of a slave based on the number of solutions sent by this slave.	+integer	10
[2] minimum_num_sendSol	Minimum number of received solutions to start the reconfiguration of a slave when such slave has not sent any solution yet.	+integer	20
[3] migration_max_time	Time in seconds (for CeSS only) which specifies when the migration stage begins.	+double	-

8.4 Problem metadata

These options set specific details for each problem. The fields with ([4]) are cases where the values of the parameters are represented by a vector given as a list of elements separated with commas. The initial point metadata is an optional field. More than one initial point can be provided.

Table 4: **Different fields of problem metadata.**

field name	description	values	default
dim	Number of decision variables.	+integer	-
neq	Number of equality constraints.	integer	0
ineq	Number of inequality constraints.	integer	0
int_var	Number of integer parameters in the problem.	integer	0
bin_var	Number of binary parameters in the problem.	integer	0
[4]lb	Lower bounds for decision variables.	+/- double	-
[4]ub	Upper bounds for decision variables.	+/- double	-
[4]cl	Lower bounds of nonlinear inequality constraints.	+/- double	-
[4]cu	Upper bounds of nonlinear inequality constraints.	+/- double	-
[4]initial point	(optional) one (or more) initial point vector. Optional fx value can be set as an attribute value.	+/- double	-

9 Setting up new problems

To set up a new problem, create the input XML file by modifying one of the templates:

(1) Edit the template file TEMPLATE_CUSTOM.xml and change the different fields as needed by the new problem. Important: set the *typebench* field to "customized" .

(2) Edit the file:

```
$  [SACESS_PATH]/benchmarks/customized/example.c
```

(3) Modify the procedure example function(double *x, void *data) in order to call the code of your new optimization problem. The **fx** value should be returned as a void pointer of the *output_function* struct. We give below a simple example (algebraic Schwefel function):

```
void* examplefunction(double *x, void *data) {

// Initialize mandatory structs

  experiment_total *exp1;
  output_function *res;
  int dim;
  exp1 = (experiment_total *) data;
  res = NULL;
  res = (output_function *) calloc(1,sizeof(output_function));
  dim = (*exp1).test.bench.dim;

// Objective function code: in this example Schwefel

  double y, sum;
  int i;

  sum = 0.0;
  for (i=0;i<dim;i++){
      sum = sum + x[i]*sin(sqrt(abs(x[i])));
  }

  y = 418.9829*dim - sum;

// Return fx: set the fx in output_function struct
// and return like a void pointer

  res->value = y;

  return res;
}
```

(4) Rebuild the library:

```
 $ make clean
 $ make all
```

10 Output: results and performance analysis

In this section, we explain the output information generated by the optimization solvers in saCeSS (note that the screen output has already been explained in in section 7).

10.1 Log file

For each processor used in an execution, a detailed log file is created in the output folder. The name of this log file follows the format logfile[id of processor]. A typical example is given below:

```
(1) +++ID[1] --- INIT ITERATION 1 - CURRENT TIME: 8.527634
(2) ID[1] - Iteration: 1 NFunEvals: 1328 Bestf: 1641518.035401 CURRENT TIME: 9.116427
(3) ID[1] - LOCAL SOLVER - Call local solver: Initial point function value:
[1641518.035401]
(4) ID[1] - LOCAL SOLVER - OUTPUT: Local solution function value: [126.305813] -
 43.105777 seconds spent in local solver - 13437 evals
(5) ID[1] - LOCAL SOLVER - Point 126.305813 is put in refset. [NEW 126.305813
 compare with OLD 1641518.035401]. LOCAL_SOLVER_TABU_LIST :  [ ***]
(6) ID[1] - REFSET FVAL - Content of the Refset in the end of iteration 1 - CURRENT
TIME 52.229497 :
 126.305813   5788249.486655   9168975.805502
 6817388.834694   11443980.798670   14009580.019673
 8449991.216290   20630016.022606   17515247.375105
 31723466.553382     10457950.384359   14739385.814609
(7) ID[1] - INDEX CHANGE COUNTER - end iteration 1 - CURRENT TIME 52.229497 :
 0   0   0   0   0   0   0   0   0   0
 0   0
(8) ID[1] - COOPERATIVE FLAG - Content of the cooperative flag iter 1 -
CURRENT TIME 52.229497 :
 0   0   0   0   0   0   0   0   0   0
 0   0
+++ID[1] - END ITERATION
```

In line (1), the solver printed the time when the iteration started in the local processor ID1. In line (2) the typical scatter search iteration output is shown, giving the number of iterations, the fbest value, the number of evaluations performed so far, and the current time. Lines (3), (4) and (5) give information about the calls to the local solver: fobj values before and after the local solver call, time spent in the local solver, or if the solution found replaces another in the Refset. Line (6) gives the content of the Refset at the end of the iteration. Line (7) shows the Index Change Counter: this vector stores the number of consecutive iterations that have not improve the solution, since eSS uses a mechanism to restart a solution above a certain threshold of this counter. Finally, line (8) indicates (for the saCeSS solver) the position in the RefSet changed by a new added element.

In the case of the saCeSS solver, the log file with ID0 will present a different output format since it plays the role of the master processor:

```
(1) ID[0] - MASTER - New solution received 2728726.138824 from processor 2
compares with current best 3376142.575050 --- CURRENT TIME 12.455868
(2) ID[0] - COMPARATION SOLUTIONS - compared candidate value 2728726.138824 with
```

```
 best know value 3376142.575050 ---> improving 19.176217 % (min 10.000000%)
  -- CURRENT TIME 12.456263 s
(3)ID[0] - MASTER ACCEPT CANDIDATE FROM 2 - fx candidate 2728726.138824 ---
CURRENT TIME 12.456922
```

The above fragment is an example of the typical messages in the master processor log. Line (1) indicates that one solution has been received in the master from slave number 2. In line (2), the **fx** of this solution is compared with the best known solution of the master. The saCeSS solver checks if the new solution satisfies the current improvement threshold (in this case a 10%). If this is fulfilled, line (3) shows that the solution is accepted by the master, and then broadcasted to the rest of the slaves.

10.2 Outputs: convergence plots

A Matlab script named **convergence.m** is generated in every run regardless of the solver used. By default this script can be used to plot the convergence curves for all processors working as slaves, appearing as black lines (see example in Figure 2). The red line represents the minimum **fx** value along the execution time. This script can be modified to obtain different plots. In addition, a csv file **convergence_id*.csv** is created for each slave in the output path with all the convergence information (for that particular slave processor).

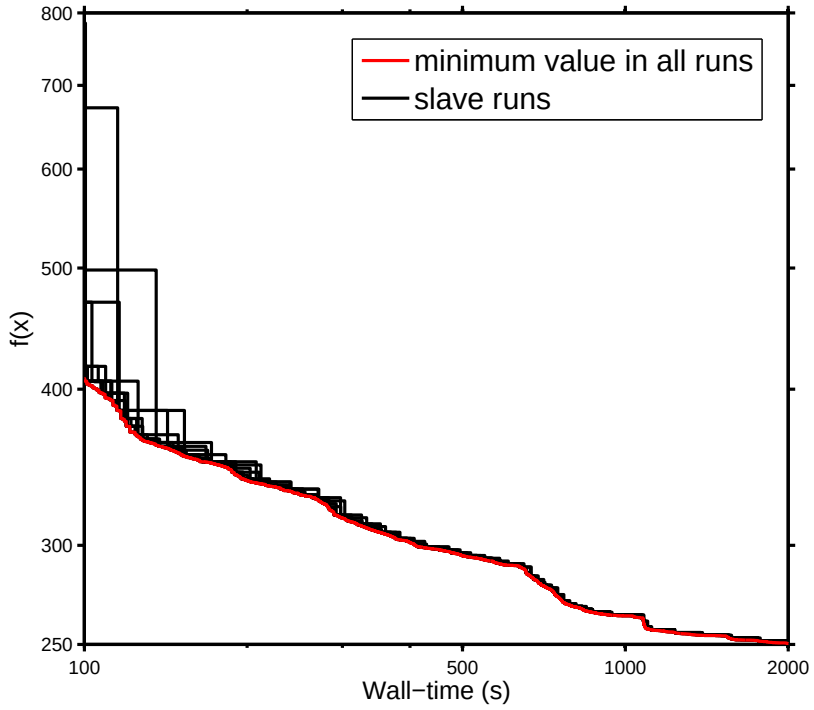

Figure 2: Convergence graph example.

10.3 Outputs: Gantt chartt

To illustrate the cooperation between processors in saCeSS, this solver generates a Matlab script gantt_id0.m, which can be used to plot a Gantt chart such as the one shown in Figure 3. The processor with tag 1 represent the master, and the rest of them are the slaves. Each red square indicates one communication, a broadcast in the case of the master and a peer to peer message in the case of the slaves. States are plotted with different colors: deep blue describes the initial evaluation phase, light blue means global search, and green corresponds to local search. This chart can be useful to analyze the algorithm cooperative behavior and detect bottlenecks in the communications, or local solver stalls. In addition, a csv fiel **gant_id*.csv** is created for each slave in the output path, containing all the numeric information presented in the Gantt chart for each processor. There are two fields in this csv: state and time. State represents if the slave is in: (1) global search, (2) local search or (3) sending a message; and time shows when the slave changed its state (in seconds).

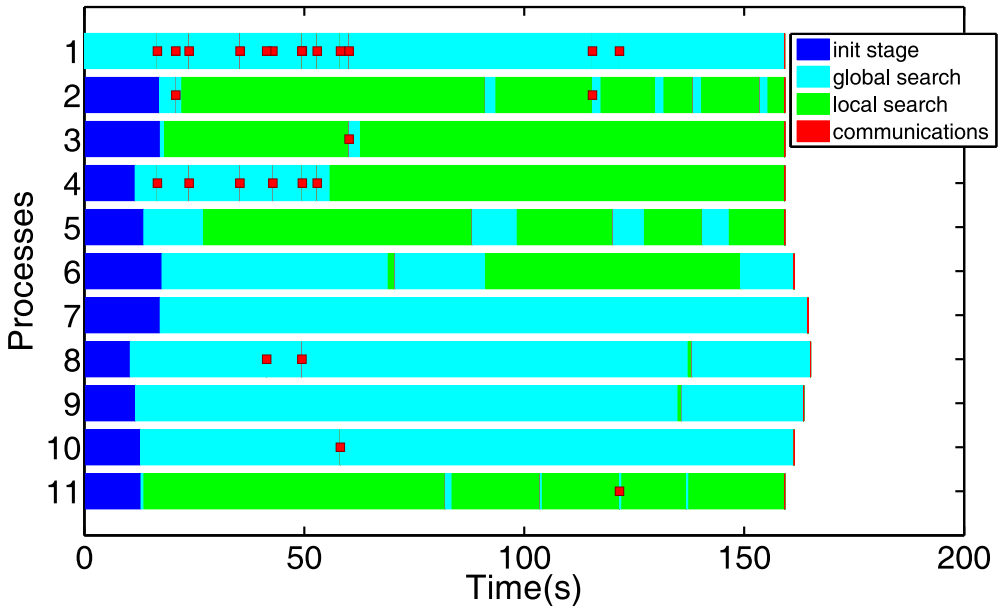

Figure 3: Gantt diagram example.

10.4 Outputs: percentage of improvement chart

saCeSS creates an additional Matlab script in the output folder, **percentage_id0.m** . Figure 4 shows a typical chart generated with this script, which illustrates the percentage of improvement of each solution shared during the cooperation. Considering that at the beginning of the execution the improvements will be notably larger than towards the end, the adaptive procedure in saCeSS starts with a large accepting threshold and decreases it as the search progress, improving the efficiency of the cooperation scheme. The initial threshold is 10%, that is, incoming solutions that improve the best known solution in the master process in at least a 10% are spread to the slaves as cooperative solutions. Once the search progresses and most of the incoming solutions are below this threshold of improvement, the master reduces the threshold to its half. This procedure is repeated, i.e. the threshold is reduced based on the incoming solutions. In addition to this script, a csv file **percentage_id0.csv** is also created, containing the corresponding data.

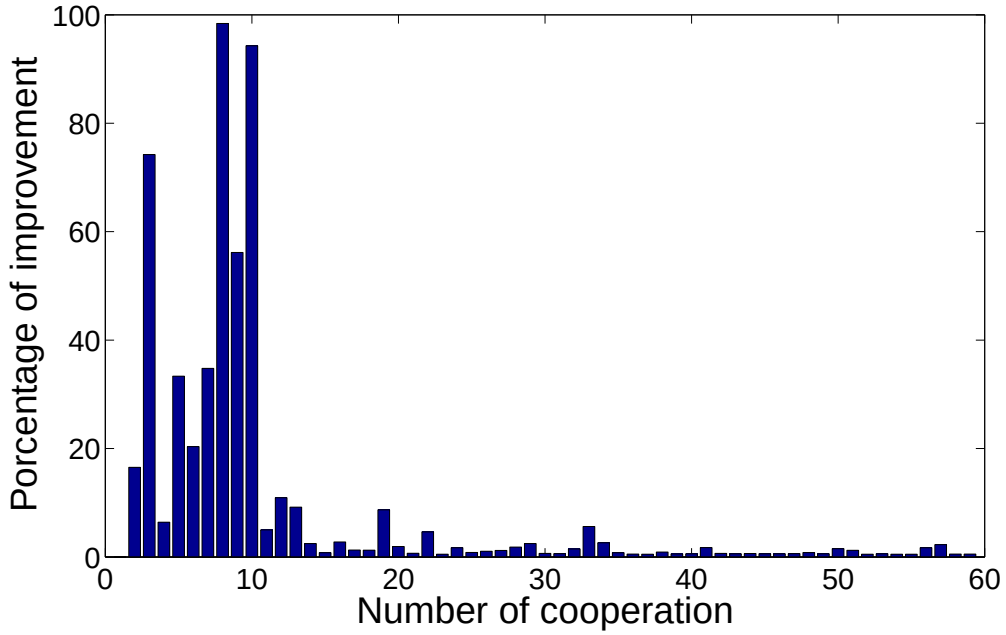

Figure 4: Percentage char example.

11 Appendix

11.1 Doxygen documentation

In addition to this document, further documentation of the code was generated with Doxygen, and can be found in the path:

```
$  [SACESS_PATH]/doc
```

To navigate it, simply change to the html subfolder, and open index.html with a browser.

11.2 Use case: solving a BBOB benchmark problem

Here we give details regarding the execution on an example from the BBOB benchmark set, using the saCeSS solver:

$$f_3(x) = 10(D - \sum_{i=1}^{D} cos(2\pi z_i)) + \|z\|^2 + f_{opt} \tag{5}$$

$$z = \Lambda^{10} T_{asy}^{0.2}(T_{osz}(x - x^{opt})) \tag{6}$$

(1) Check the XML file TEMPLATE_BBOB.xml.

```
<xml>
   <run>
      <typebench> noiselessBBOB </typebench>
      <id> 3 </id>
      <log_scale> 1 </log_scale>
      <output>  1 </output>
      <verbose> 1 </verbose>
      <local_search> 1 </local_search>
      <stopping_criteria>
         <maxevaluation> 1e10 </maxevaluation>
         <maxtime> 1e10 </maxtime>
         <vtr> -462.0899999900 </vtr>
      </stopping_criteria>
   </run>
   <method name="saCeSS">
      <user_options>
        <weight> default </weight>
        <tolc> default </tolc>
        <prob_bound>  default </prob_bound>
        <nstuck_solution> default  </nstuck_solution>
      </user_options>
      <global_options>
        <dim_ref> default </dim_ref>
        <ndiverse> default </ndiverse>
        <combination> default </combination>
        <n_stuck> default </n_stuck>
      </global_options>
      <local_options>
        <solver> dhc </solver>
        <tol> 2 </tol>
        <evalmax> 10000 </evalmax>
        <iterprint> 0 </iterprint>
        <n1> 1 </n1>
        <n2> 10 </n2>
        <balance> 0.25 </balance>
        <bestx> default </bestx>
```

```
      </local_options>
   </method>
   <parallelization name="cooperative">
      <reception_threshold> 1 </reception_threshold>
      <evals_threshold> 1000 </evals_threshold>
      <mult_num_sendSol> 10 </mult_num_sendSol>
      <minimum_num_sendSol> 20 </minimum_num_sendSol>
      <migration_max_time> 10 </migration_max_time>
   </parallelization>
   <problem>
      <dim> 100 </dim>
      <neq> 0 </neq>
      <ineq> 0 </ineq>
      <int_var> 0 </int_var>
      <bin_var> 0 </bin_var>
      <lb>
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,
      -5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0,-5d0
      </lb>
      <ub>
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,
      5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0,5d0
      </ub>
   </problem>
</xml>
```

(2) The above input file corresponds to an example of a noiseless BBOB problem. The id field of the file is equal to 3 (corresponding to BBOB problem Rastrigin function with monotone transformation separable "condition" 10*. The code of the objective function is:

```
TwoDoubles f3BBOB(double* x) {
  /* Rastrigin with monotone transformation separable "condition" 10*/
  int i, rseed; /*Loop over dim*/
```

```
  static unsigned int funcId = 3;
  static double condition = 10.;
  static double beta = 0.2;
  double tmp, tmp2, Fadd, Fval, Ftrue = 0.;
  TwoDoubles res;

  if (!isInitDone)
  {
  rseed = funcId + 10000 * trialid;
  /*INITIALIZATION*/
  Fopt = computeFopt(funcId, trialid);
  computeXopt(rseed, DIM);
  isInitDone = 1;
  }

  Fadd = Fopt;
  for (i = 0; i < DIM; i++)
  {
  tmx[i] = x[i] - Xopt[i];
  }

  monotoneTFosc(tmx);
  for (i = 0; i < DIM; i++)
  {
  tmp = ((double)i)/((double)(DIM-1));
  if (tmx[i] > 0)
  tmx[i] = powl(tmx[i], 1 + beta * tmp * sqrtl(tmx[i]));
  tmx[i] = powl(sqrtl(condition), tmp) * tmx[i];
  }
  /* COMPUTATION core*/
  tmp = 0.;
  tmp2 = 0.;
  for (i = 0; i < DIM; i++)
  {
  tmp += cosl(2*M_PI*tmx[i]);
  tmp2 += tmx[i]*tmx[i];
  }
  Ftrue = 10 * (DIM - tmp) + tmp2;
  Ftrue += Fadd;
  Fval = Ftrue; /* without noise*/

  res.Fval = Fval;
  res.Ftrue = Ftrue;
  return res;
}
```

(3) Rebuild the library:

```
$ make clean
$ make all
```

(4) Execute the following command to start the optimization with saCeSS and 10 slave processors (plus the master processor, i.e. total of 11 processors):

```
 $ mpirun -np 11 bin/paralleltestbed  inputs/TEMPLATE_BBOB.xml output/TEST
```

(5) An example of the output is:

```
=====================================================
== PARALLEL SACESS OPTIMIZATION LIBRARY =============
=====================================================

************ GENERAL INFORMATION ********************

* SOLVER :: ScatterSearch
Version solver : saCeSS - SELF-ADAPTED ASYNCHONOUS COOPERATIVE eSS
Topology       : mixture between master-slave and ring topology
Max.Evals      : 10000000000.0
Max.Time(s)    : 10000000000.0
Local Solver   : dhc

* TOTAL PROCESSORS USED :: 11
Number of MPI proc.                : 11
Number of openMP threads per proc. : 1

* TRANSLATION ENABLED: logarithmic space is enabled and there are negative values in the bounds
* TYPE OF BENCKMARK :: noiselessBBOB
Name                     : bbob noiseless benchmark ID=3

Number of parameters     : 100
Value To Reach           : -462.0899999900
[DEFAULT] Value To Reach : -462.0899999900

************ SLAVE INFORMATION *********************
ID SLAVE:  10        Refset size:   8      ndiverse: 1000
ID SLAVE:   1        Refset size:  11      ndiverse: 1000
ID SLAVE:   2        Refset size:  18      ndiverse: 1000
ID SLAVE:   3        Refset size:  23      ndiverse: 1000
ID SLAVE:   4        Refset size:  33      ndiverse: 1000
ID SLAVE:   5        Refset size:  40      ndiverse: 1000
ID SLAVE:   6        Refset size:  36      ndiverse: 1000
ID SLAVE:   7        Refset size:  28      ndiverse: 1000
ID SLAVE:   8        Refset size:  23      ndiverse: 1000
ID SLAVE:   9        Refset size:  15      ndiverse: 1000

************ OPTIMIZATION BEGINS ******************

[PROCESSOR ID=  5] Initial Pop: NFunEvals: 1005 Bestf: 0.3589258222D+04 CPUTime: 0.08
[PROCESSOR ID=  7] Initial Pop: NFunEvals: 1005 Bestf: 0.3708624027D+04 CPUTime: 0.08
[PROCESSOR ID=  6] Initial Pop: NFunEvals: 1005 Bestf: 0.3409052147D+04 CPUTime: 0.08
[PROCESSOR ID=  9] Initial Pop: NFunEvals: 1005 Bestf: 0.3506803831D+04 CPUTime: 0.08
[PROCESSOR ID=  1] Initial Pop: NFunEvals: 1005 Bestf: 0.3472608934D+04 CPUTime: 0.08
```

```
[PROCESSOR ID=  3] Initial Pop: NFunEvals: 1005 Bestf: 0.3404969664D+04 CPUTime: 0.08
[PROCESSOR ID=  2] Initial Pop: NFunEvals: 1005 Bestf: 0.3077374332D+04 CPUTime: 0.08
[PROCESSOR ID= 10] Initial Pop: NFunEvals: 1005 Bestf: 0.3232373437D+04 CPUTime: 0.08
[PROCESSOR ID=  4] Initial Pop: NFunEvals: 1005 Bestf: 0.3429437440D+04 CPUTime: 0.08
[PROCESSOR ID=  8] Initial Pop: NFunEvals: 1005 Bestf: 0.3277398695D+04 CPUTime: 0.08
[MASTER] Best Known Solution fx = -162.0560129956 -- CURRENT TIME 0.566901
[MASTER] Best Known Solution fx = -171.9258142017 -- CURRENT TIME 0.747273
[MASTER] Best Known Solution fx = -220.7454870828 -- CURRENT TIME 1.041798
[MASTER] Best Known Solution fx = -242.6345409584 -- CURRENT TIME 1.792014
[MASTER] Best Known Solution fx = -268.5032791053 -- CURRENT TIME 2.132476
[MASTER] Best Known Solution fx = -272.4831104024 -- CURRENT TIME 2.544211
[MASTER] Best Known Solution fx = -278.4769139796 -- CURRENT TIME 2.608388
[MASTER] Best Known Solution fx = -315.8648315468 -- CURRENT TIME 2.925781
[MASTER] Best Known Solution fx = -316.3330524539 -- CURRENT TIME 4.426304
[MASTER] Best Known Solution fx = -316.9540051528 -- CURRENT TIME 4.515316
[MASTER] Best Known Solution fx = -317.1321474232 -- CURRENT TIME 4.699846
[MASTER] Best Known Solution fx = -318.8497036769 -- CURRENT TIME 4.971063
[MASTER] Best Known Solution fx = -319.8446627361 -- CURRENT TIME 4.987311
[MASTER] Best Known Solution fx = -320.0751089210 -- CURRENT TIME 5.179142
[MASTER] Best Known Solution fx = -320.4303622174 -- CURRENT TIME 5.292093
[MASTER] Best Known Solution fx = -322.2558032112 -- CURRENT TIME 5.426295
[MASTER] Best Known Solution fx = -323.4788379560 -- CURRENT TIME 5.493517
[MASTER] Best Known Solution fx = -325.3209000200 -- CURRENT TIME 5.560635
[MASTER] Best Known Solution fx = -325.4819980692 -- CURRENT TIME 5.627924
[MASTER] Best Known Solution fx = -329.0377225150 -- CURRENT TIME 5.695135
[MASTER] Best Known Solution fx = -330.7556063864 -- CURRENT TIME 6.245887
[MASTER] Best Known Solution fx = -331.7505654434 -- CURRENT TIME 6.618156
[MASTER] Best Known Solution fx = -332.1449949266 -- CURRENT TIME 7.929244

...

[MASTER] Best Known Solution fx = -460.5010667497 -- CURRENT TIME 181.678443
[MASTER] Best Known Solution fx = -460.6572128384 -- CURRENT TIME 181.900189
[MASTER] Best Known Solution fx = -460.7312508943 -- CURRENT TIME 181.988984
[MASTER] Best Known Solution fx = -460.8168400790 -- CURRENT TIME 182.120092
[MASTER] Best Known Solution fx = -460.8340698219 -- CURRENT TIME 182.257591
[MASTER] Best Known Solution fx = -461.0950409428 -- CURRENT TIME 182.269358
[MASTER] Best Known Solution fx = -461.1708805487 -- CURRENT TIME 251.564004
[SLAVE ID=1] fx = -462.0899999999 < VTR =-462.0899999900 -- CURRENT TIME 251.961785 s
[MASTER] Best Known Solution fx = -462.0899999999 -- CURRENT TIME 251.962177
[SLAVE ID=4] fx = -462.0899999969 < VTR =-462.0899999900 -- CURRENT TIME 251.964223 s

************ RESULTS ******************************

[PROGRAM FINISHED] Desired function value achieved.

ID   6  -- SEED RANDOM NUMBERS:   128284329262089.0
ID   0  -- SEED RANDOM NUMBERS:   18317570460189.0
ID   2  -- SEED RANDOM NUMBERS:   54961473672105.0
ID   9  -- SEED RANDOM NUMBERS:   183336346613420.0
```

```
ID   3  -- SEED RANDOM NUMBERS:   73287806423832.0
ID   1  -- SEED RANDOM NUMBERS:   36638061684224.0
ID  10  -- SEED RANDOM NUMBERS:   201702109677068.0
ID   8  -- SEED RANDOM NUMBERS:   164976425077464.0
ID   7  -- SEED RANDOM NUMBERS:   146622345069200.0
ID   5  -- SEED RANDOM NUMBERS:   109949234218824.0
ID   4  -- SEED RANDOM NUMBERS:   91617059939405.0

BEST SOLUTION:
-2.340800023775449822949212830281
2.300000045786584124130058626179
2.213599976910797906271000101697
0.728000025009398754605172143783
2.299200055741467352277140889782
2.175199960564798473683367774356
2.547200009837631284881354076788
2.598400021828865646966733038425
1.803200064147326919794522837037
0.057599930004560206953101442195
-2.750400025023791883427293214481
-3.784000048040497610202237410704
-0.138399961925948389307450270280
-2.960800064440693546430338756181
-1.384800078019431168740993598476
-3.924000041733659571718817460351
-0.339200000560775372093758051051
2.948000024838817623162867676001
-2.829599964024567704257151490310
1.819999921426228794985036074650
-0.614400006121983643936346197734
3.398399979991030050996414502151
-2.630400058521315642678928270470
2.006399996169097477149989572354
-2.064000019009911035539062140742
-1.462399980229230678219209949020
0.572000017054391740600749471923
-2.877599924146208465458585123997
2.599200073073655303801388072316
3.341600005557159747127116133925
-0.251999989691819692438912170473
-1.953599956087260558490470430115
-1.864800048135854204645056597656
1.193600011497295554363518022001
0.983199932469746684660094615538
-3.102399947850253525416519551072
-1.732000029679864994136551104020
3.513599945032503057973372051492
1.863999960069286032648960826918
-2.041600074660798735948219473357
-2.010400073576127155661197321024
```

1.251199932244830392846779432148
-1.546400051574575940094291581772
3.919999990266196832067180366721
-3.984799951829604225395087269135
1.066400000245554480216014781035
-3.495200020973226351372886711033
-1.008000000574703847178170690313
-2.068000028269861179097688363981
-3.761600052174424391182583349291
-1.417600072883336892459738010075
-2.052000062317573370052059544832
-0.897599936422567523663929023314
-2.789599960860485783342710419674
-0.114400021048277622526256891433
3.144799982263350557332159951329
1.775200064156866375242316280492
-1.152000015578761527024198585423
0.616799986977079051087002881104
0.808800035140282425061286630807
2.811999949288527034241269575432
2.468799964855477924174920190126
-3.385599967045680003252527967561
-1.630400063950943945201288443059
0.946399963073170980010218045209
1.212799933268793317608924553497
-0.369599933227155119652707071509
0.591199961670084483955633913865
-2.461599973667238039354288048344
-0.396000065695213621097536815796
1.093600046736815656345243041869
-0.284800081578397090709131589392
-0.211199983324844353660409979057
-1.171999960313474531403699074872
3.844800065488861662288400111720
-3.391999969314944962661684257910
2.495999923591124769473026390187
3.882399981144841305535919673275
3.377600033934854195649677421898
0.583999997857564068226565723307
-3.186400010168878527139213474584
0.979200056507415261819460283732
-1.895999969821241748491047474090
3.326399989129951606514623563271
1.311200006642979865034703834681
1.796800004541710293892720073927
1.047199974719114301535682898248
-2.629600005290166908622495611780
-1.707999981719780180355883203447
1.287199959528980031109313131310
-1.397600000526855623661504068878

```
-2.902400024323041893836716553778
3.272800049044899850514411809854
2.747999948291210969841813493986
1.695199963320789038334623910487
-3.960800002365496119693943910534
1.186399966293296692754211107967
2.621600065628996389932581223547
-1.564800030368790118018296197988
-1.216800014371416160940952977398
f(x)=-462.089999999870258307055337354541

TOTAL TIME TO REACH VTR : 251.961785
TOTAL EVALUATIONS       : 32048193
AVERAGE ITERATIONS      : 2149.700000
EVALUATION FAILED       : 0

f-3 DIM 100 fbest: -462.08999999987025830706 < ftarget -462.08999999000002389948
numevals (32048193) -- time VTR -> 251.961785 -- iter 2149.700000

************ GRAPHS GENERATED **********************

GRAPH PATH               : output/TEST
CONVERGENCE GRAPH MATLAB : output/TEST/convergence.m
CONVERGENCE GRAPHs CSV   : output/TEST/convergence_id*.csv
GANTT CHART MATLAB       : output/TEST/gantt_id0.m
GANTT CHARTs CSV         : output/TEST/gantt_id*.csv
PERCENTAGE GRAPH MATLAB  : output/TEST/percentage_id0.m
PERCENTAGE GRAPH CSV     : output/TEST/percentage_id0.csv
```

(6) The solver also created three plots (as Matlab scripts) in the path output/TEST : Figure 5, Figure 6 and Figure 7.

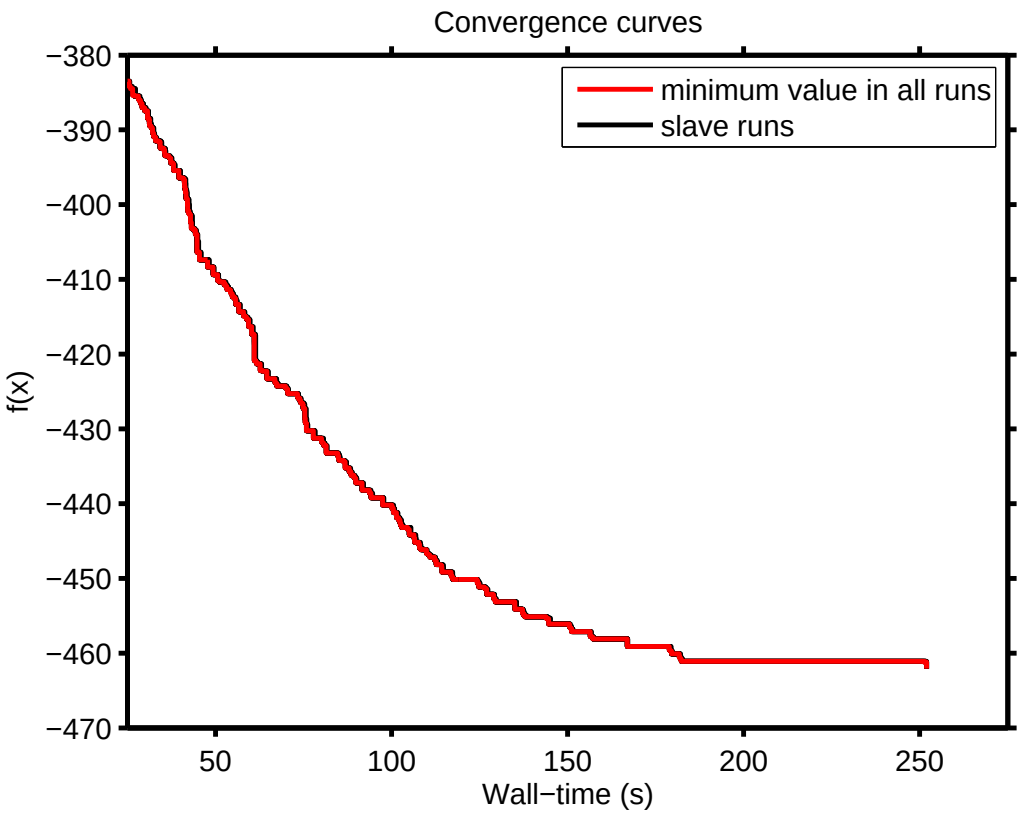

Figure 5: Convergence curves.

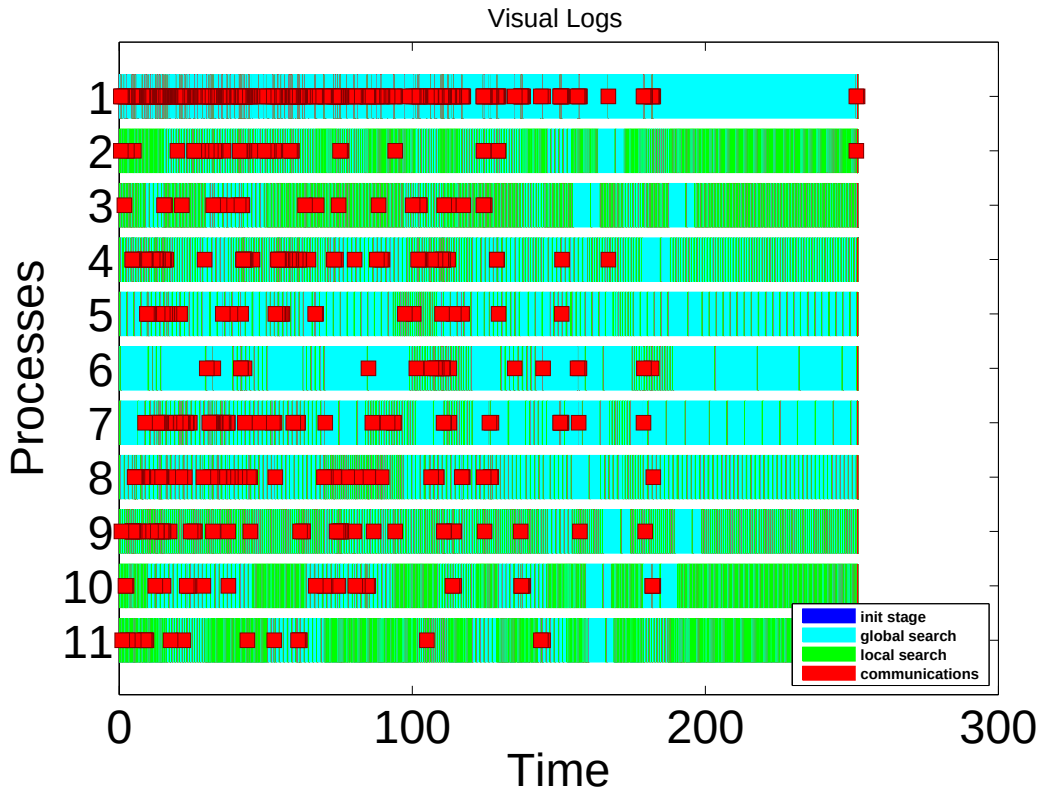

Figure 6: Gantt chart.

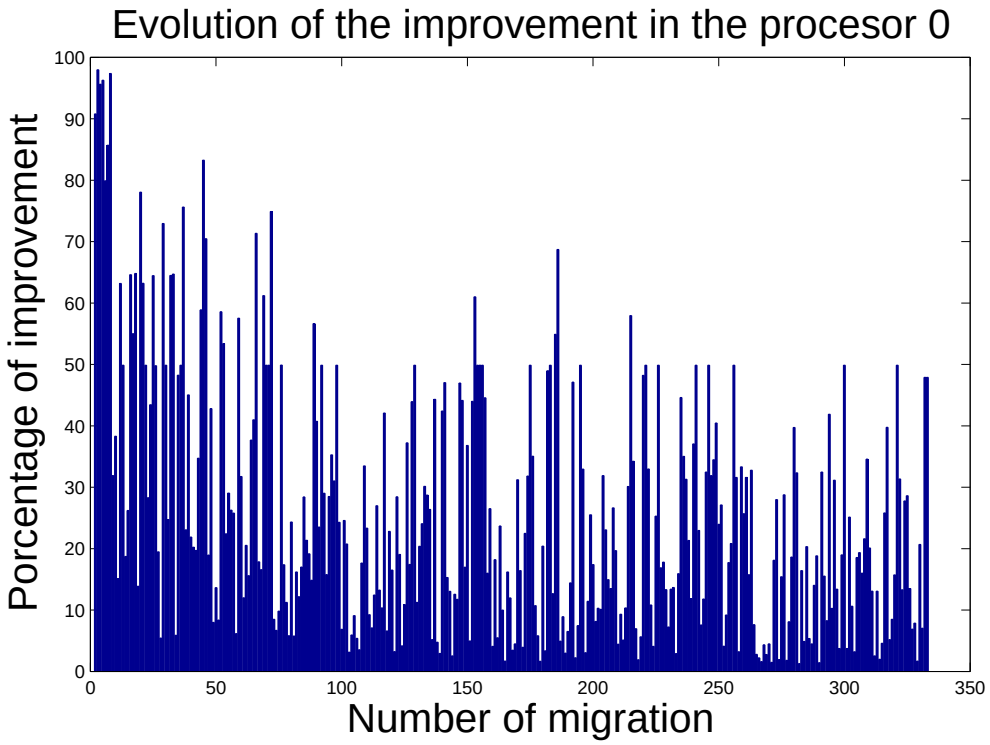

Figure 7: Bar chart of the percentage of improvement of each spread solution.

11.3 Use case: setting up and running a new problem

To create and run a new problem with the saCeSS library:

(1) Create a new XML input file (you can use the template file TEMPLATE_CUSTOM.xml). Set the different options and input data. Important: set the *typebench* field to "customized" .

```
<xml>
    <run>
        <typebench> customized </typebench>
        <id> 0 </id>
        <log_scale> 1 </log_scale>
        <output>  1 </output>
        <verbose> 1 </verbose>
        <local_search> 1 </local_search>
        <stopping_criteria>
           <maxevaluation> 1e10 </maxevaluation>
           <maxtime> 1e10 </maxtime>
           <vtr> 0 </vtr>
        </stopping_criteria>
    </run>
    <method name="saCeSS">
        <user_options>
            <weight> default </weight>
            <tolc> default </tolc>
            <prob_bound>  default </prob_bound>
            <nstuck_solution> default  </nstuck_solution>
        </user_options>
        <global_options>
            <dim_ref> default </dim_ref>
            <ndiverse> default </ndiverse>
            <combination> default </combination>
            <n_stuck> default </n_stuck>
        </global_options>
        <local_options>
            <solver> dhc </solver>
            <tol> 2 </tol>
            <evalmax> 5000 </evalmax>
            <iterprint> 0 </iterprint>
            <n1> 1 </n1>
            <n2> 10 </n2>
            <balance> 0.25 </balance>
            <bestx> default </bestx>
        </local_options>
    </method>
    <parallelization name="cooperative">
        <reception_threshold> 1 </reception_threshold>
        <evals_threshold> 1000000 </evals_threshold>
        <mult_num_sendSol> 10 </mult_num_sendSol>
        <minimum_num_sendSol> 20 </minimum_num_sendSol>
```

```
        <migration_max_time> 10 </migration_max_time>
    </parallelization>
    <problem>
       <dim> 100 </dim>
       <neq> 0 </neq>
       <ineq> 0 </ineq>
       <int_var> 0 </int_var>
       <bin_var> 0 </bin_var>
       <lb>
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,
          -500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0,-500d0
        </lb>
        <ub>
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,
          500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0,500d0
        </ub>
    </problem>
</xml>
```

(2) Edit the file:

```
$  [SACESS_PATH]/benchmarks/customized/example.c
```

(3) This file has a procedure called examplefunction(double *x, void *data) where the user should include the code (or call to other code) corresponding to the new optimization problems. The fx value should be returned as a void pointer of the *output_function* struct.

```
void* examplefunction(double *x, void *data) {

// Initialize mandatory structs

experiment_total *exp1;
output_function *res; // result struct
int dim; // dimension of the problem
```

```
exp1 = (experiment_total *) data; // main struct of the library
res = NULL;
res = (output_function *) calloc(1,sizeof(output_function));
dim = (*exp1).test.bench.dim;

// Objective function code: in this example Schwefel

double y, sum;
int i;

sum = 0.0;
for (i=0;i<dim;i++){
sum = sum + x[i]*sin(sqrt(abs(x[i])));
}

y = 418.9829*dim - sum;

// Return fx: set the fx in output_function struct
// and return like a void pointer

res->value = y;

return res;
}
```

(4) Rebuild the library:

```
 $ make clean
 $ make all
```

(5) Run the following command to start optimization with saCeSS and 10 processors as slaves:

```
$ mpirun -np 11 bin/paralleltestbed  inputs/TEMPLATE_CUSTOM.xml output/TEST_CUSTOM
```

(6) An example of the output is:

```
=====================================================
== PARALLEL SACESS OPTIMIZATION LIBRARY =============
=====================================================

************ GENERAL INFORMATION ********************

* SOLVER :: ScatterSearch
Version solver : saCeSS - SELF-ADAPTED ASYNCHONOUS COOPERATIVE eSS
Topology       : mixture between master-slave and ring topology
Max.Evals      : 10000000000.0
Max.Time(s)    : 10000000000.0
Local Solver   : none

* TOTAL PROCESSORS USED :: 11
Number of MPI proc.                : 11
Number of openMP threads per proc. : 1
```

```
* TYPE OF BENCKMARK :: customized
Name                      : user benchmarks
Number of parameters      : 100
Value To Reach            : 0.0000000000
[DEFAULT] Value To Reach : 0.0000000000

************ SLAVE INFORMATION ********************
ID SLAVE:   7        Refset size:  28      ndiverse: 1000
ID SLAVE:   8        Refset size:  23      ndiverse: 1000
ID SLAVE:   1        Refset size:  11      ndiverse: 1000
ID SLAVE:   3        Refset size:  23      ndiverse: 1000
ID SLAVE:   4        Refset size:  33      ndiverse: 1000
ID SLAVE:   6        Refset size:  36      ndiverse: 1000
ID SLAVE:   2        Refset size:  18      ndiverse: 1000
ID SLAVE:   9        Refset size:  15      ndiverse: 1000
ID SLAVE:  10        Refset size:   8      ndiverse: 1000
ID SLAVE:   5        Refset size:  40      ndiverse: 1000

************ OPTIMIZATION BEGINS ******************

[PROCESSOR ID=6 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3332957946D+05 CPUTime: 0.02
[PROCESSOR ID=3 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3173251090D+05 CPUTime: 0.02
[PROCESSOR ID=4 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3239093801D+05 CPUTime: 0.02
[PROCESSOR ID=7 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3355700655D+05 CPUTime: 0.02
[PROCESSOR ID=8 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3112890956D+05 CPUTime: 0.03
[PROCESSOR ID=10] Initial Pop: NFunEvals: 1005 Bestf: 0.2903213049D+05 CPUTime: 0.03
[PROCESSOR ID=5 ] Initial Pop: NFunEvals: 1005 Bestf: 0.2911745468D+05 CPUTime: 0.03
[PROCESSOR ID=1 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3268108196D+05 CPUTime: 0.03
[PROCESSOR ID=2 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3112469732D+05 CPUTime: 0.03
[PROCESSOR ID=9 ] Initial Pop: NFunEvals: 1005 Bestf: 0.3014593713D+05 CPUTime: 0.03
[MASTER] Best Known Solution fx = 27871.1210540254 -- CURRENT TIME 0.032514
[MASTER] Best Known Solution fx = 24920.9797588601 -- CURRENT TIME 0.036967
[MASTER] Best Known Solution fx = 21161.3641607849 -- CURRENT TIME 0.044651
[MASTER] Best Known Solution fx = 18162.4175535570 -- CURRENT TIME 0.048600
[MASTER] Best Known Solution fx = 15998.2879337977 -- CURRENT TIME 0.060139
[MASTER] Best Known Solution fx = 14748.3012905604 -- CURRENT TIME 0.061979
[MASTER] Best Known Solution fx = 13491.0670517032 -- CURRENT TIME 0.067884
[MASTER] Best Known Solution fx = 12765.4429053550 -- CURRENT TIME 0.072879
[MASTER] Best Known Solution fx = 11966.9488069243 -- CURRENT TIME 0.076049
[MASTER] Best Known Solution fx = 11240.1171200323 -- CURRENT TIME 0.079383
[MASTER] Best Known Solution fx = 10568.1831157165 -- CURRENT TIME 0.082457
[MASTER] Best Known Solution fx = 10074.9738863525 -- CURRENT TIME 0.086643
[MASTER] Best Known Solution fx = 9743.2614332453 -- CURRENT TIME 0.088121
[MASTER] Best Known Solution fx = 9486.8022418830 -- CURRENT TIME 0.093640
[MASTER] Best Known Solution fx = 9216.7715317187 -- CURRENT TIME 0.098087
[MASTER] Best Known Solution fx = 8874.6077973916 -- CURRENT TIME 0.102544
[MASTER] Best Known Solution fx = 8610.0426777809 -- CURRENT TIME 0.105719
[MASTER] Best Known Solution fx = 8277.9447876973 -- CURRENT TIME 0.108560
[MASTER] Best Known Solution fx = 8167.6387034070 -- CURRENT TIME 0.111710
```

```
[MASTER] Best Known Solution fx = 7908.3826073462 -- CURRENT TIME 0.112664
[MASTER] Best Known Solution fx = 7564.2904712111 -- CURRENT TIME 0.114514
[MASTER] Best Known Solution fx = 7367.0202278710 -- CURRENT TIME 0.116384
[MASTER] Best Known Solution fx = 7217.1994850730 -- CURRENT TIME 0.119600
[MASTER] Best Known Solution fx = 7060.4875896525 -- CURRENT TIME 0.121017
[MASTER] Best Known Solution fx = 6928.8891522983 -- CURRENT TIME 0.123918
[MASTER] Best Known Solution fx = 6816.8394144221 -- CURRENT TIME 0.126788
[MASTER] Best Known Solution fx = 6709.9447931591 -- CURRENT TIME 0.127071
[MASTER] Best Known Solution fx = 6623.9468136574 -- CURRENT TIME 0.130966
[MASTER] Best Known Solution fx = 6495.3309382312 -- CURRENT TIME 0.132890
[MASTER] Best Known Solution fx = 6216.7990733657 -- CURRENT TIME 0.135369
[MASTER] Best Known Solution fx = 5978.6381134837 -- CURRENT TIME 0.137103
[MASTER] Best Known Solution fx = 5779.0369343600 -- CURRENT TIME 0.138791

...

[MASTER] Best Known Solution fx = 134.6405024079 -- CURRENT TIME 4.594338
[MASTER] Best Known Solution fx = 133.9269260015 -- CURRENT TIME 4.754363
[MASTER] Best Known Solution fx = 115.9185255446 -- CURRENT TIME 4.801592
[MASTER] Best Known Solution fx = 89.8546821397 -- CURRENT TIME 4.803052
[MASTER] Best Known Solution fx = 50.3931553410 -- CURRENT TIME 4.804483
[SLAVE ID=9] fx = -22.2085115321 < VTR =0.0000000000 -- CURRENT TIME 4.805871 s
[SLAVE ID=3] fx = -32.2503405799 < VTR =0.0000000000 -- CURRENT TIME 4.807059 s

************ RESULTS ******************************

[PROGRAM FINISHED] Desired function value achieved.

ID   0  -- SEED RANDOM NUMBERS:         94098609046582.0
ID   7  -- SEED RANDOM NUMBERS:        752870797748608.0
ID   3  -- SEED RANDOM NUMBERS:        376411991624032.0
ID   5  -- SEED RANDOM NUMBERS:        564635542873752.0
ID   6  -- SEED RANDOM NUMBERS:        658751707358038.0
ID   8  -- SEED RANDOM NUMBERS:        847005980623740.0
ID   9  -- SEED RANDOM NUMBERS:        941132385780020.0
ID  10  -- SEED RANDOM NUMBERS:       1035293901811708.0
ID   4  -- SEED RANDOM NUMBERS:        470522304295750.0
ID   1  -- SEED RANDOM NUMBERS:        188200143999448.0
ID   2  -- SEED RANDOM NUMBERS:        282304604858598.0

BEST SOLUTION:
421.76111294111450433774734848 2476
421.79905625374954070139210671 1864
422.01865053142284978093812242 1503
422.39892185622971965131000615 6564
419.93128976669845542346592992 5442
417.97204030843289501717663370 0728
421.95440801197452174164936877 7871
422.67155745445830916651175357 4014
421.57453854241362023458350449 8005
```

421.99420277678285629008314572274 7
421.81499712300256987873581238091 0
422.86667411207963596098124980926 5
425.77370408543026769621064886450 8
422.33813793707616923711611889302 7
422.97297728120838655740953981876 4
422.02617514988907032602583058178 4
422.16744032130020514159696176648 1
421.07640331048213511166977696120 7
423.88913145600531606760341674089 4
420.80260541895489723174250684678 6
421.99976694748789896038942970335 5
420.41318541555392584996297955513 0
422.89672052626218601290020160377 0
421.99444125815102779597509652376 2
424.30862937056826922344043850898 7
421.70482510701691580834449268877 5
422.77219173910123117821058258414 3
421.50195469242135004606097936630 2
424.14670934297220128428307361900 8
420.93660265375575590951484628021 7
421.29843265012806341474060900509 4
422.53423831601156734905089251697 1
421.93261752814191822835709899663 9
422.91860066649326199694769456982 6
421.90712136845758095660130493342 9
422.00571722645622685377020388841 6
421.65321669259566306209308095276 4
422.28936950641616476787021383643 2
422.01488403790051506803138181567 2
420.82910202826224121963605284690 9
419.99437362169339849060634151101 1
419.63401914899509392853360623121 3
422.90816102954829602822428569197 7
420.59904781432828713150229305028 9
421.04074529237675506010418757796 3
422.00564351656890949016087688505 6
421.80889837453804602773743681609 6
422.96476481745332876016618683934 2
422.92021118686182035162346437573 4
422.95352792384915119328070431947 7
422.98239514886626011502812616527 1
420.89805805621955414608237333595 8
420.88906412592575634334934875369 1
412.05196880584946939052315428853 0
421.19285166374316986548365093767 6
421.96001623370813149449531920254 2
421.79437648952449535499908961355 7
420.92814853607990244199754670262 3
422.07211482364130006317282095551 5

```
420.83064654347680288992705754935 7
421.99851828959077693070867098867 9
420.77680808537598977636662311852 0
421.57784958363606619968777522444 7
420.91212001253046537385671399533 7
422.88554272413608714487054385244 8
421.88716748783940602152142673730 9
421.97220200616197871568147093057 6
417.54146102688332575780805200338 4
421.87525792710471250757109373807 9
422.02166851154402138490695506334 3
422.72731020846703131610411219298 8
422.98339312966282932393369264900 7
421.99908976923524051017011515796 2
416.85820798091572214616462588310 2
422.82635855627802357048494741320 6
422.90367891965996705039287917316 0
421.71086641115965676362975500524 0
423.09922595348427876160712912678 7
420.94203591698203581472625955939 3
422.30453762036177067784592509269 7
420.89126506120328485849313437938 7
422.94731586149163149457308463752 3
422.68688594072608566420967690646 6
424.03940138401060266915010288357 7
423.96323094845968171284766867756 8
422.25136237821936902037123218178 7
422.05846026817118854523869231343 3
421.96751321606990359214250929653 6
421.84811664196200808873982168734 1
422.96680646937568326393375173211 1
421.94477514664845330116804689168 9
423.08056509080608975637005642056 5
423.99345330428951683643390424549 6
420.68779433872987283393740653991 7
422.02956273640455719942110590636 7
421.95518052656166219094302505254 7
422.91599457583652110770344734191 9
422.96487004672900411605951376259 3
420.11551076503252488691941834986 2
422.84384355940215982627705670893 2
f(x)=-32.250340579870680812746286392212

TOTAL TIME TO REACH VTR : 4.805871
TOTAL EVALUATIONS       : 1910347
AVERAGE ITERATIONS      : 1986.100000
EVALUATION FAILED       : 0

f-0 DIM 100 fbest: -32.25034057987068081275 < ftarget 0.00000000000000000000 numevals (1910347) --
```

```
************ GRAPHS GENERATED **********************

GRAPH PATH               : output/TEST_CUSTOM
CONVERGENCE GRAPH MATLAB : output/TEST_CUSTOM/convergence.m
CONVERGENCE GRAPHs CSV   : output/TEST_CUSTOM/convergence_id*.csv
GANTT CHART MATLAB       : output/TEST_CUSTOM/gantt_id0.m
GANTT CHARTs CSV         : output/TEST_CUSTOM/gantt_id*.csv
PERCENTAGE GRAPH MATLAB  : output/TEST_CUSTOM/percentage_id0.m
PERCENTAGE GRAPH CSV     : output/TEST_CUSTOM/percentage_id0.csv
```

11.4 Use case: solving a problem defined in AMIGO with saCeSS

AMIGO (Advanced model Identification using GLobal Optimization) is a Matlab toolbox that can be helpful for defining and solving parameter estimation problems, particularly those defined by models based on ordinary differential equations. AMIGO allows the use of a number of sequential optimization solvers (including eSS). However, for complex (i.e. large and/orand computationally expensive) problems, users might want to use a parallel solver, such as saCeSS.

The latest version of AMIGO (AMIGO2, available at https://sites.google.com/site/amigo2toolbox/home) allows users to, for a given problem, export C code which can then be used with saCeSS. In fact, the BioPreDyn-benchmark problems included with saCeSS have been created using this new feature. Below we describe the steps that must be followed, using a simple example (the classical alfa-pinene problem) to illustrate the procedure. It should be noted that this example is small and obviously does not require the use of a parallel solver (it can be solved in seconds using a sequential optimizer), but it is used below for the sake of simplicity.

Important: basic knowledge of AMIGO, Matlab and Linux is required.

First of all, install and check AMIGO2. Then, follow these steps:

(1) Select the AMIGO input file (Matlab) that you want to export. This file must be a valid AMIGO input (i.e. make sure it runs OK in AMIGO before attempting the export). Let us consider the toy example alpha_pinene_PE.m in the folder:

```
$  [SACESS\_PATH]/examples
```

(1.1) Check the content of the AMIGO input file and make sure it can be processed by AMIGO:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TITLE: Thermal isomerization of alfa-pinene
% REF:   R.E. Fuguitt y J. E. Hawkins, 1947. Rate of thermal isomerization
% of ?-pinene in the liquid phase. J. A. C. S., 69:461
%
% NOTE!!!: [] indicates that the corresponding input may be omitted,
%              default value will be assigned
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%============================
% RESULTS PATHS RELATED DATA
%============================
% Folder to keep results (in Results) for a given problem
% To identify figures and reports for a given problem
% ADVICE: the user may introduce any names related to the problem at hand
% [] Identifier required in order not to overwrite previous results
%    This may be modified from command line. 'run1'(default)
inputs.pathd.results_folder='alpha_pinene';
inputs.pathd.short_name='pinene';
inputs.pathd.runident='r1';
%============================
% MODEL RELATED DATA
%============================
% Model introduction: 'charmodelC'|'c_model'|'charmodelM'|'matlabmodel'|'sbmlmodel'|
% 'blackboxmodel'|'blackboxcost
inputs.model.input_model_type='charmodelC';
inputs.model.n_st=5;       % Number of states
```

```
inputs.model.n_par=5;      % Number of model parameters
inputs.model.n_stimulus=0; % Number of inputs, stimuli or control variables
% [] Names given to states/pars/inputs: 'standard' (x1,x2,...p1,
% p2...,u1, u2,...) 'custom'(default)
inputs.model.names_type='standard';
% Equations describing system dynamics. Time derivatives are regarded
% 'd'st_name''
inputs.model.eqns=...
char('dx1=-(p1+p2)*x1',...
'dx2= p1*x1',...
'dx3= p2*x1-(p3+p4)*x3+p5*x5',...
'dx4= p3*x3',...
'dx5= p4*x3-p5*x5');

p1=5.93e-5;  p2=2.96e-5;  p3=2.05e-5;  p4=27.5e-5;  p5=4e-5;
% Nominal value for the parameters, this allows to fix known parameters
% These values may be updated during optimization
inputs.model.par=[p1 p2 p3 p4 p5];

%==================================
% EXPERIMENTAL SCHEME RELATED DATA
%==================================
inputs.exps.n_exp=1;       % Number of experiments
inputs.exps.n_obs{1}=4;    % Number of observed quantities per experiment
inputs.exps.obs{1}=char('y1=x1','y2=x2','y3=x3','y4=x5');
% Initial conditions for each experiment
inputs.exps.exp_y0{1}=[100 0 0 0 0];
inputs.exps.t_f{1}=36420;  % Experiments duration
inputs.exps.n_s{1}=8;     % Number of sampling times
% [] Sampling times, by default equidistant
inputs.exps.t_s{1}=[1230 3060 4920 7800 10680 15030 22620 36420];

%==================================
% EXPERIMENTAL DATA RELATED INFO
%==================================
inputs.exps.data_type='real'; % Type of data: 'pseudo'|'pseudo_pos'|'real'
inputs.exps.noise_type='homo';
inputs.exps.std_dev{1}=[0.001 0.001 0.001 0.001];
%Experimental data 1:
inputs.exps.exp_data{1}=[
88.35 7.3   2.3  1.75
76.4  15.6  4.5  2.8
65.1  23.1  5.3  5.8
50.4  32.9  6.0  9.3
37.5  42.7  6.0  12.0
25.9  49.1  5.9  17.0
14.0  57.4  5.1  21.0
4.5   63.1  3.8  25.7
];
```

```
%==================================
% UNKNOWNS RELATED DATA
%==================================
% GLOBAL UNKNOWNS (SAME VALUE FOR ALL EXPERIMENTS)
% 'all'|User selected
inputs.PEsol.id_global_theta='all';
% Maximum allowed values for the paramters
inputs.PEsol.global_theta_max=[1 1 1 1 1];
% Minimum allowed values for the parameters
inputs.PEsol.global_theta_min= [0 0 0 0 0];

% % GLOBAL INITIAL CONDITIONS
% [] 'all'|User selected| 'none' (default)
inputs.PEsol.id_global_theta_y0='x1';
% Maximum allowed values for the initial conditions
inputs.PEsol.global_theta_y0_max=[110];
% Minimum allowed values for the initial conditions
inputs.PEsol.global_theta_y0_min=[90];

%==================================
% COST FUNCTION RELATED DATA
%==================================
% 'lsq' (weighted least squares default) | 'llk' (log likelihood)
%               | 'user_PEcost'
inputs.PEsol.PEcost_type='lsq';
inputs.PEsol.lsq_type='Q_I';

%==================================
% NUMERICAL METHODS RELATED DATA
%==================================
% SIMULATION
% [] IVP solver: 'radau5'(default, fortran)|'rkf45'|'lsodes'|
inputs.ivpsol.ivpsolver='cvodes';
% [] Sensitivities solver: 'cvodes' (C)
inputs.ivpsol.senssolver='cvodes';

inputs.ivpsol.rtol=1.0D-7;   % [] IVP solver integration tolerances
inputs.ivpsol.atol=1.0D-7;
```

(1.2) Add the the following definitions to the input file (which are used to generate the necessary C-files for saCeSS):

```
% Model type must be 'charmodelC' to use C
inputs.model.input_model_type='charmodelC';
% Generates gcc command
inputs.model.exe_type='fullC';
% Indicates the name of the generated C-files
inputs.model.odes_file=fullfile(pwd,'amigoRHS.c');
```

(1.3) Use AMIGO_Prep to process the resulting input file. This will create a C-file amigoRHS.c, and two Matlab structures (inputs and privstruct).

```
[inputs privstruct]=AMIGO_Prep(inputs);
```

(1.4) Save those structures in a MAT-file compatible with saCeSS using:

```
% This line is very important to generate a .MAT file compatible with saCeSS
save('load_C','-v7.3','inputs','privstruct');
```

(2) Make sure the the C-file amigoRHS.c and the M-file load_C.mat have been created.

(3) Move the generated files into the saCeSS benchmarks folder creating a new subfolder, such as e.g.:

```
$  cd [SACESS_PATH]/benchmarks/systemsBiology/others
$  mkdir alpha-pinene
$  mv [PATH_OF_GENERATED_MATLAB_CODE]/load_C.mat  alpha-pinene
$  mv [PATH_OF_GENERATED_MATLAB_CODE]/amigoRHS.c  alpha-pinene
```

(4) Rename amigoRHS.c as amigoRHS_alpha_pinene.c and create amigoRHS_alpha_pinene.h. Copy the code of amigoRHS_alpha_pinene.c into amigoRHS_alpha_pinene.h using:

```
$  cd [SACESS_PATH]/benchmarks/systemsBiology/others/alpha-pinene
$  mv amigoRHS.c  amigoRHS_alpha_pinene.c
$  cp amigoRHS_alpha_pinene.c amigoRHS_alpha_pinene.h
```

(5) Modify amigoRHS_alpha_pinene.h to create a correct header-file. Delete all code AFTER the main(argc,argv) function (included this). Delete also the line $\#include < amigoRHS.h >$. Besides, you can delete all the code between $\#if$ and $\#endif$ for the definition of the function run_amigo_main(), because it is not necesary for the saCeSS library. In addition, rename the functions:

```
amigoRHS()
amigoRHS_get_OBS()
amigoRHS_get_sens_OBS()
amigo_Y_at_tcon()
```

with the name of the new problem, in this case:

```
amigoRHS_alpha_pinene()
amigoRHS_get_OBS_alpha_pinene()
amigoRHS_get_sens_OBS_alpha_pinene()
amigo_Y_at_tcon_alpha_pinene()
```

obtaining the following file:

```
$  cat amigoRHS_alpha_pinene.h

int amigoRHS_alpha_pinene(realtype t, N_Vector y, N_Vector ydot, void *data);
void amigoRHS_get_OBS_alpha_pinene(void* data);
void amigoRHS_get_sens_OBS_alpha_pinene(void* data);
void amigo_Y_at_tcon(void* data,realtype t, N_Vector y);
```

(6) Modify amigoRHS_alpha_pinene.c to create a correct C-file. Delete all code BEFORE the main function (included this). Replace the following includes:

```
#include <amigoRHS.h>
#include <amigoJAC.h>
#include <amigoSensRHS.h>
```

by the AMIGO_model include:

```
#include <AMIGO_model.h>
```

Besides, rename the functions:

```
amigoRHS()
amigoRHS_get_OBS()
amigoRHS_get_sens_OBS()
amigo_Y_at_tcon()
```

with the name of the new problem, e.g.:

```
amigoRHS_alpha_pinene()
amigoRHS_get_OBS_alpha_pinene()
amigoRHS_get_sens_OBS_alpha_pinene()
amigo_Y_at_tcon_alpha_pinene()
```

obtaining the following file:

```
cat amigoRHS_alpha_pinene.c

#include <math.h>
#include <AMIGO_model.h>

/**** Definition of the states *** */
#define x1 Ith(y,0)
#define x2 Ith(y,1)
#define x3 Ith(y,2)
#define x4 Ith(y,3)
#define x5 Ith(y,4)
#define iexp amigo_model->exp_num

/* *** Definition of the sates derivative *** */
#define dx1 Ith(ydot,0)
#define dx2 Ith(ydot,1)
#define dx3 Ith(ydot,2)
#define dx4 Ith(ydot,3)
#define dx5 Ith(ydot,4)

/* *** Definition of the parameters *** */
#define p1 (*amigo_model).pars[0]
#define p2 (*amigo_model).pars[1]
#define p3 (*amigo_model).pars[2]
#define p4 (*amigo_model).pars[3]
#define p5 (*amigo_model).pars[4]

/* *** Definition of the algebraic variables *** */
/* Right hand side of the system (f(t,x,p))*/
```

```
int amigoRHS_alpha_pinene(realtype t, N_Vector y, N_Vector ydot,
     void *data){
     AMIGO_model* amigo_model=(AMIGO_model*)data;

/* *** Equations *** */

     dx1=-(p1+p2)*x1;
     dx2=p1*x1;
     dx3=p2*x1-(p3+p4)*x3+p5*x5;
     dx4=p3*x3;
     dx5=p4*x3-p5*x5;

     return(0);

}

/* Jacobian of the system (dfdx)*/
int amigoJAC(long int N, realtype t, N_Vector y, N_Vector fy, DlsMat J,
     void *user_data, N_Vector tmp1, N_Vector tmp2, N_Vector tmp3){

     AMIGO_model* amigo_model=(AMIGO_model*)user_data;

     return(0);
}

/* R.H.S of the sensitivity dsi/dt = (df/dx)*si + df/dp_i */
int amigoSensRHS(int Ns, realtype t, N_Vector y, N_Vector ydot, int iS,
     N_Vector yS, N_Vector ySdot, void *data, N_Vector tmp1, N_Vector tmp2){

     AMIGO_model* amigo_model=(AMIGO_model*)data;

     return(0);

}

#define x1 (amigo_model->sim_results[0][j])
#define x2 (amigo_model->sim_results[1][j])
#define x3 (amigo_model->sim_results[2][j])
#define x4 (amigo_model->sim_results[3][j])
#define x5 (amigo_model->sim_results[4][j])

void amigoRHS_get_OBS_alpha_pinene(void* data){
     int j;
     double t;
     AMIGO_model* amigo_model=(AMIGO_model*)data;

     switch (amigo_model->exp_num){
     #define y1 amigo_model->obs_results[0][j]
     #define y2 amigo_model->obs_results[1][j]
     #define y3 amigo_model->obs_results[2][j]
```

```
     #define y4 amigo_model->obs_results[3][j]

          case 0:
          for (j = 0; j < amigo_model->n_times; ++j){
               y1=x1;
               y2=x2;
               y3=x3;
               y4=x5;
          }
          break;
     }
}

#define x1 (amigo_model->sens_results[0][j][k])
#define x2 (amigo_model->sens_results[1][j][k])
#define x3 (amigo_model->sens_results[2][j][k])
#define x4 (amigo_model->sens_results[3][j][k])
#define x5 (amigo_model->sens_results[4][j][k])

void amigoRHS_get_sens_OBS_alpha_pinene(void* data){
     int j,k;

     AMIGO_model* amigo_model=(AMIGO_model*)data;
     switch (amigo_model->exp_num){

     case 0:
     #define y1 amigo_model->sens_obs[0][j][k]
     #define y2 amigo_model->sens_obs[1][j][k]
     #define y3 amigo_model->sens_obs[2][j][k]
     #define y4 amigo_model->sens_obs[3][j][k]

     for (j = 0; j < amigo_model->n_total_x; ++j){
          for (k = 0; k < amigo_model->n_times; ++k){
               y1=x1;
               y2=x2;
               y3=x3;
               y4=x5;
          }
     }
     break;
     }
}


void amigo_Y_at_tcon_alpha_pinene(void* data,realtype t, N_Vector y){
     AMIGO_model* amigo_model=(AMIGO_model*)data;

}
```

(7) Add the C files and the header-files created in the previous steps to the Makefile:

```
$ cat Makefile
...
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/3-step_pathway/*.c)
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/circadian/*.c)
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/Nfkb/*.c)
...
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/circadian/
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/Nfkb/
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/3-step_pathway/
...
```

Define the new folder with the problem files (e.g. alpha-pinene) assigning the following variables:

```
$ cat Makefile
...
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/alpha-pinene/*.c)
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/3-step_pathway/*.c)
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/circadian/*.c)
SRC+=$(wildcard $(BUILDDIR)/benchmarks/systemsBiology/others/Nfkb/*.c)
...
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/alpha-pinene/
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/circadian/
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/Nfkb/
INC+=-I$(BUILDDIR)/benchmarks/systemsBiology/others/3-step_pathway/
```

(8) Edit and modify the file *benchmark_functions_SystemBiology.c* to add the C-file and h-file previously generated to the saCeSS library:

```
$ edit benchmarks/systemsBiology/benchmark_functions_SystemBiology.c
```

(8.1) Add the include of the new header-file:

```
#include <amigoRHS_alpha_pinene.h> // INCLUDE NEW HEADER
#include <amigoRHS_CIRCADIAN.h>
#include <amigoRHS_3step.h>
#include <amigoRHS_NFKB.h>
#include <amigoRHS_B1.h>
```

(8.2) Add the path where the files of the new AMIGO benchmark are stored and assign a ID NUMBER BENCHMARK in the function:

```
int load_benchmark_SystemBiology(experiment_total *exp) {
     ...
}
```

Add a new ID value (we will use 11 here) for the alpha pinene problem. Define there the path of the MAT-file and assign the same ID value to the variable *type*. The struct values jf and VTR_default are only shown in the execution (the real VTR to be used will be defined in the input XML file).

```
int load_benchmark_SystemBiology(experiment_total *exp) {
 ...
 if (exp->test.bench.current_bench == 0) {
    path = "benchmarks/systemsBiology/others/circadian/load_C.mat";
    type = 0; // SYSTEMS BIOLOGY ID 0
```

```
    (*exp).test.VTR_default = 1e-5; // ONLY TO SHOW IN THE SCREEN
    (*exp).test.jf = 1e-5; // ONLY TO SHOW IN THE SCREEN
 }
 ...
 // ADD A NEW CONDITION - ALPHA PINENE - ID BENCHMARK 11
 else if (exp->test.bench.current_bench == 11) {
    path = "benchmarks/systemsBiology/others/alpha_pinene/load_C.mat";
    type = 11; // SYSTEMS BIOLOGY ID 11
    (*exp).test.jf = 0; // ONLY TO SHOW IN THE SCREEN
    (*exp).test.VTR_default = 0; // ONLY TO SHOW IN THE SCREEN
 }
 ...
}
```

(8.3) In the same function *load_benchmark_SystemBiology*(), locate the conditional where the openMatFileAMIGO(path) function extracts the options of the M-file. Add there a new condition setting the variable "*type*" to the ID number, 11 in our example (set this using SYSTEMS BIOLOGY ID). Complete the functions set_AMIGO_problem_rhs() and set_AMIGO_problem_obs_function() inside the condition with the functions previously defined:

```
amigoRHS_alpha_pinene()
amigoRHS_get_OBS_alpha_pinene()
amigoRHS_get_sens_OBS_alpha_pinene()
amigo_Y_at_tcon_alpha_pinene()

int load_benchmark_SystemBiology(experiment_total *exp) {
...
 // OPEN MAT CONDITIONAL
 if (noamigo != 1){
  // EXTRACT THE OPTIONS OF THE M-FILE
  exp->amigo =  openMatFileAMIGO(path);

  if (type == 0) {
      set_AMIGO_problem_rhs(exp->amigo, amigoRHS_CIRCADIAN,
                      amigo_Y_at_tcon_CIRCADIAN);
      set_AMIGO_problem_obs_function(exp->amigo, amigoRHS_get_OBS_CIRCADIAN,
                      amigoRHS_get_sens_OBS_CIRCADIAN);
  }
  ...
  // ADD NEW CONDITION - PROBLEM ALPHA PINENE
  else if (type == 11) {
      set_AMIGO_problem_rhs(exp->amigo, amigoRHS_alpha_pinene,
             amigo_Y_at_tcon_alpha_pinene);
      set_AMIGO_problem_obs_function(exp->amigo, amigoRHS_get_OBS_alpha_pinene,
             amigoRHS_get_sens_OBS_alpha_pinene);
  }
  ...
 }
...
} // END OF load_benchmark_SystemBiology
```

(8.4) OPTIONAL: modify the following functions to return the name of the new benchmark problem in the output screen:

```
const char* return_benchmark_SystemBiology(int i){
...
   } else if (i == 11) {
    return "alpha pinene";
   }
...
}

char * getnameSB(int id){
...
    else if (id == 11) name = "alpha pinene";
...
}
```

(9) Increase the value of $MAXNUMSB$ in file *setup_benchmarks.c*. In our example, the value 10 increases to 11:

```
$ edit [SACESS_PATH]/src/setup_benchmarks.c

#define MINNUMSB 0
#define MAXNUMSB 11
```

(10) Recompile the code with the commands:

```
$  make clean
$  make all
```

(11) Create a new XML input file (you can use as template another problem in Systems Biology such as TEMPLATE_CIRCADIAN.xml). Set the different options and input data. Important: set the typebench field to "systemBiology" and the "id" to 11. The following is an example of the XML input file:

```
$ edit TEMPLATE_alpha_pinene.xml

<xml>
  <run>
    <typebench> systemBiology </typebench>
    <id> 11 </id>
    <log_scale> 0 </log_scale>
    <output>  1 </output>
    <verbose> 1 </verbose>
    <local_search> 1 </local_search>
    <maxtime> 1e10 </maxtime>
    <stopping_criteria>
        <maxevaluation> 1e10 </maxevaluation>
        <maxtime> 1e10 </maxtime>
        <vtr> 17.1722  </vtr>
    </stopping_criteria>
  </run>
```

```
  <method name="saCeSS">
        <user_options>
            <weight> default </weight>
            <tolc> default </tolc>
            <prob_bound>  default </prob_bound>
            <nstuck_solution> default  </nstuck_solution>
        </user_options>
        <global_options>
            <dim_ref> default </dim_ref>
            <ndiverse> default </ndiverse>
            <combination> default </combination>
            <n_stuck> -1 </n_stuck>
        </global_options>
        <local_options>
            <solver> nl2sol </solver>
            <tol> 2 </tol>
            <evalmax> 10000 </evalmax>
            <iterprint> 0 </iterprint>
            <n1> 1 </n1>
            <n2> 10 </n2>
            <balance> 0.25 </balance>
            <bestx> default </bestx>
        </local_options>
  </method>
    <parallelization name="cooperative">
        <reception_threshold> 1 </reception_threshold>
        <evals_threshold> 5000 </evals_threshold>
        <mult_num_sendSol> 10 </mult_num_sendSol>
        <minimum_num_sendSol> 20 </minimum_num_sendSol>
        <migration_max_time> 10 </migration_max_time>
    </parallelization>
    <problem>
        <dim> 5 </dim>
        <neq> 0 </neq>
        <ineq> 0 </ineq>
        <int_var> 0 </int_var>
        <bin_var> 0 </bin_var>
        <ub> 1.0,1.0,1.0,1.0,1.0 </ub>
        <lb> 0.0,0.0,0.0,0.0,0.0 </lb>
    </problem>
</xml>
```

(12) Execute the problem (see previous instructions in this manual for details regarding execution options), e.g.:.

```
$  mpirun -np 2 bin/paralleltestbed inputs/TEMPLATE_ALPHA_PINENE.xml output/TEST
```

11.5 Use case: solving constrained nonlinear programming (cNLP) problems

The saCeSS library can handle constrained problems (containing both inequality and equality constraints). The following toy example from the MEIGO toolbox [19] is used to illustrate the process of solving constrained optimization problems with saCeSS:

$$\min_x f(x) = -x_4 \tag{7}$$

subject to

$$x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0 \tag{8}$$

$$x_1 - 1 + k_1 x_1 x_5 = 0 \tag{9}$$

$$x_2 - x_1 - 1 k_2 x_2 x_6 = 0 \tag{10}$$

$$x_3 + x_1 - 1 + k_3 x_3 x_5 = 0 \tag{11}$$

$$x_5^{0.5} + x_6^{0.5} \leq 4 \tag{12}$$

$$0 \leq x_1, x_2, x_3, x_4 \leq 1 \tag{13}$$

$$0 \leq x_5, x_6 \leq 16 \tag{14}$$

with $k_1 = 0.09755988$, $k_3 = 0.0391908$, $k_2 = 0.99 k_1$ and $k_4 = 0.9 k_3$.

Please note that this problem is very simple and therefore it does not make sense to use a parallel method to solve it. We are using it purely for the sake of simplicity in the explanations.

(1) Check the following Matlab code from the MEIGO toolbox defining this problem (i.e., *ex3.m*):

```
$ cat example/ex3.m // file with objective function

function [f,g]=ex3(x,k1,k2,k3,k4)

f=-x(4);
%Equality constraints
g(1)=x(4)-x(3)+x(2)-x(1)+k4*x(4).*x(6);
g(2)=x(1)-1+k1*x(1).*x(5);
g(3)=x(2)-x(1)+k2*x(2).*x(6);
g(4)=x(3)+x(1)-1+k3*x(3).*x(5);
%Inequality constraint
g(5)=x(5).^0.5+x(6).^0.5;

return

$ cat examples/main_ex3.m

%====================== PROBLEM SPECIFICATIONS ==============
problem.f='ex3';
problem.x_L=[0 0 0 0 0 0];
problem.x_U=[1 1 1 1 16 16];
problem.neq=4;
problem.c_L=-inf;
```

```
problem.c_U=4;

opts.maxtime=5;
opts.local.solver='solnp';
%================ END OF PROBLEM SPECIFICATIONS ==============
k1=0.09755988;
k3=0.0391908;
k2=0.99*k1;
k4=0.9*k3;

[Results]=MEIGO(problem,opts,'ESS',k1,k2,k3,k4);
```

(2) In order to solve the same problem with saCeSS, the objective function and the constraints must be defined in C within the file:
[SACESS_PATH]/examples/constraints_example.c.

```
$ cat constraints_example.c

void* examplefunction(double *x, void *data) {
  experiment_total *exp1;
  output_function *res;
  double F, dim, *g;
  double k1,k2,k3,k4;

  k1=0.09755988;
  k3=0.0391908;
  k2=0.99*k1;
  k4=0.9*k3;

  exp1 = (experiment_total *) data;

//  DIMENSION
  dim = 6;

//  SET THE VARS
  res = NULL;
  res = (output_function *) calloc(1,sizeof(output_function));
  g = (double *) calloc(4,sizeof(output_function));

//  f=-x(4);
  F = -x[3];

//  Equality constraints :
//  g(1)=x(4)-x(3)+x(2)-x(1)+k4*x(4).*x(6);
  g[0]=x[3]-x[2]+x[1]-x[0]+k4*x[3] *x[5];

//  g(2)=x(1)-1+k1*x(1).*x(5);
  g[1]=x[0]-1+k1*x[0]*x[4];

//  g(3)=x(2)-x(1)+k2*x(2).*x(6);
```

```
  g[2]=x[1]-x[0]+k2*x[1] *x[5];

//  g(4)=x(3)+x(1)-1+k3*x(3).*x(5);
  g[3]=x[2]+x[0]-1+k3*x[2] *x[4];

// Inequality constraint
// g(5)=x(5).^0.5    +    x(6).^0.5;
  g[4]=pow(x[4],0.5)+pow(x[5],0.5);

//  PUT IN THE output_function
  res->value = F;
  res->g = g;

  return res;
}
```

(3) Now apply the same steps as in section 9 to define the new problem. To complete the definition of the problem in saCeSS, we also need an XML input file such as:

```
$ cat inputs/TEMPLATE_CONSTRAINTS_EXAMPLE.xml

<xml>
  <run>
    <typebench> customized </typebench>
    <id> 0 </id>
    <log_scale> 0 </log_scale>
    <output>  1 </output>
    <verbose> 1 </verbose>
    <local_search> 0 </local_search>
    <stopping_criteria>
        <maxevaluation> 1e10 </maxevaluation>
        <maxtime> 2 </maxtime>
        <vtr> -0.388811 </vtr>
    </stopping_criteria>
  </run>
  <method name="saCeSS">
    <user_options>
      <weight> default </weight>
      <tolc> default </tolc>
      <prob_bound>  default </prob_bound>
      <nstuck_solution> default  </nstuck_solution>
    </user_options>
    <global_options>
      <dim_ref> default </dim_ref>
      <ndiverse> default </ndiverse>
      <combination> default </combination>
      <n_stuck> default </n_stuck>
    </global_options>
    <local_options>
      <solver> dhc </solver>
      <tol> 2 </tol>
```

```
      <evalmax> 5000 </evalmax>
      <iterprint> 0 </iterprint>
      <n1> 1 </n1>
      <n2> 10 </n2>
      <balance> 0.25 </balance>
      <bestx> default </bestx>
    </local_options>
  </method>
  <parallelization name="cooperative">
    <reception_threshold> 1 </reception_threshold>
    <evals_threshold> 5000 </evals_threshold>
    <mult_num_sendSol> 10 </mult_num_sendSol>
    <minimum_num_sendSol> 20 </minimum_num_sendSol>
    <migration_max_time> 10 </migration_max_time>
  </parallelization>
  <problem>
    <dim> 6 </dim>
    <ineq> 1 </ineq>
    <neq> 4 </neq>
    <int_var> 0 </int_var>
    <bin_var> 0 </bin_var>
    <lb> 0,0,0,0,0,0</lb>
    <ub> 1,1,1,1,16,16 </ub>
    <cu> 4 </cu>
    <cl> -inf </cl>
  </problem>
</xml>
```

(4) Note that in the above input file, the following fields define the number and bounds for the constraints:

```
// Number of equality constraints
<neq> 4 </neq>
// Number of inequality constraints
<ineq> 1 </ineq>
// Lower bounds of nonlinear inequality constraints
<cu> 4 </cu>
// Upper bounds of nonlinear inequality constraints
<cl> -inf </cl>
```

(6) Now, recompile the code and run it like in the previous sections:

```
$ make clean
$ make all
$ mpirun -np 4 bin/paralleltestbed inputs/TEMPLATE_CONSTRAINTS_EXAMPLE.xml output/TEST
```

11.6 Use case: solving mixed-integer non-linear programming (MINLP) problems

The saCeSS library can solve mixed-integer non-linear programming (MINLP) problems. The process to define this type of problems is similar to the ones described above, but make sure that in the C-based problem definition:

(i) integer and binary variables must be declared as C integer types inside the objective function.

(ii) integer and binary variables decision variables are stored at the end of the decision variables vector.

(iii) in the XML input file, the field "*int_var*" is used for integer variables, and "*bin_var*" for binary variables. These fields must contain the NUMBER of integer and binary variables respectively.

Below, we describe the steps to define and run a very simple MINLP problem, re-using another example from the MEIGO toolbox. [19]:

$$\min_x f(x) = x_2^2 + x_3^2 + 2x_1^2 + x_4^2 - 5x_2 - 5x_3 - 21x_1 + 7x_4 \tag{15}$$

subject to

$$x_2^2 + x_3^2 + x_1^2 + x_4^2 + x_2 - x_3 + x_1 - x_4 \leq 8 \tag{16}$$

$$x_2^2 + 2x_3^2 + x_1^2 + 2x_4^2 - x_2 - x_4 \leq 10 \tag{17}$$

$$2x_2^2 + x_3^2 + x_1^2 + 2x_2 - x_3 - x_4 \leq 5 \tag{18}$$

$$0 \geq x_i \geq 10 \quad \forall i \in [1, .., 4] \tag{19}$$

Integer variables: x_2, x_3 and x_4.

Again, please note that this problem is very simple and therefore it does not really make sense to use a parallel method to solve it. We are using it purely for the sake of simplicity in the explanations.

(1) Check the Matlab code of MEIGO toolbox for this problem (i.e., *ex4.m*):

```
$ cat examples/ex4.m // file with objective function
function [F,g]=ex4(x)
F = x(2)^2 + x(3)^2 + 2.0*x(1)^2 + x(4)^2 - 5.0*x(2) - 5.0*x(3) - 21.0*x(1) + 7.0*x(4);
g(1) = x(2)^2 + x(3)^2 + x(1)^2 + x(4)^2 + x(2) - x(3) + x(1) - x(4);
g(2) = x(2)^2 + 2.0*x(3)^2 + x(1)^2 + 2.0*x(4)^2 - x(2) - x(4);
g(3) = 2.0*x(2)^2 + x(3)^2 + x(1)^2 + 2.0*x(2) - x(3) - x(4);
return

$ cat examples/main_ex4.m // file to configure and run the problem

%======================== PROBLEM SPECIFICATIONS ===========================
problem.f='ex4';
problem.x_L=[0 0 0 0];
problem.x_U=[10 10 10 10];
```

```
problem.int_var=3;
problem.c_L=[-inf -inf -inf ];
problem.c_U=[8 10 5];
problem.x_0=[3 4 5 1 ];


opts.local.solver=0;
opts.maxtime=2;
%======================== END OF PROBLEM SPECIFICATIONS ====================

Results=MEIGO(problem,opts,'ESS');
```

(2) To implement the same problem for saCeSS, we need to wrote a C-code version. Check the one provided in *examples/minlp_constraints_example.c*. In this file, the integer variables are declared with a type integer and stored at the end of the vector of decision variables. Please note that it is very important to store the values F (value of the objective function) and g (values of the constraints) in the corresponding fields of the C-struct $output\_function *res$.

```
#include <structure_paralleltestbed.h>

void* examplefunction(double *x, void *data) {
  experiment_total *exp1;
  output_function *res;
  double x1, F, dim, *g;
  int x2,x3,x4;

  exp1 = (experiment_total *) data;

//  DIMENSION
  dim = 4;

//  SET THE VARS
  x1 = x[0];
  x2 = (int) x[1];
  x3 = (int) x[2];
  x4 = (int) x[3];

  res = NULL;
  res = (output_function *) calloc(1,sizeof(output_function));
  g = (double *) calloc(3,sizeof(double));

//  F = x(2)^2 + x(3)^2 + 2.0*x(1)^2 + x(4)^2 - 5.0*x(2)
//      - 5.0*x(3) - 21.0*x(1) + 7.0*x(4);
  F = (double) integer_power(x2,2);
  F = F + ((double)integer_power(x3,2));
  F = F + 2.0*pow(x1,2.0);
  F = F + ((double)integer_power(x4,2));
  F = F - ((double) 5*x2);
  F = F - ((double) 5*x3);
  F = F - 21.0*x1;
```

```
  F = F + ((double)7*x4);

//  CONSTRAINTS :
//  g(1) = x(2)^2 + x(3)^2 + x(1)^2 + x(4)^2 + x(2) - x(3) + x(1) - x(4);
  g[0]= (double) integer_power(x2,2);
  g[0]= g[0] + ((double) integer_power(x3,2)) + pow(x1,2.0);
  g[0]= g[0] + ((double) integer_power(x4,2));
  g[0]= g[0] + ((double) x2) - ((double) x3);
  g[0]= g[0] + x1 - ((double) x4);

//  g(2) = x(2)^2 + 2.0*x(3)^2 + x(1)^2 + 2.0*x(4)^2 - x(2) - x(4);
  g[1]= (double) integer_power(x2,2);
  g[1]= (double) integer_power(x2,2);
  g[1]=g[1]+((double) 2*integer_power(x3,2));
  g[1]=g[1]+  pow(x1,2.0);
  g[1]=g[1]+((double) 2*integer_power(x4,2));
  g[1]=g[1]+((double) - x2 - x4 );

//  g(3) = 2.0*x(2)^2 + x(3)^2 + x(1)^2 + 2.0*x(2) - x(3) - x(4);
  g[2]= (double) 2 * integer_power(x2,2);
  g[2]= g[2] + integer_power(x3,2);
  g[2]= g[2] + pow(x1,2.0);
  g[2]= g[2] + ((double)  2*x2);
  g[2]= g[2] - x3 -x4;

// PUT IN THE output_function
  res->value = F;
  res->g = g;

  return res;
}
```

(3) Follow the same steps as in section 9 to define the XML input file:

```
<xml>
  <run>
    <typebench> customized </typebench>
    <id> 0 </id>
    <log_scale> 0 </log_scale>
    <output>  1 </output>
    <verbose> 1 </verbose>
    <local_search> 0 </local_search>
    <stopping_criteria>
        <maxevaluation> 1e10 </maxevaluation>
        <maxtime> 2 </maxtime>
        <vtr> -0.388811 </vtr>
    </stopping_criteria>
  </run>
  <method name="saCeSS">
    <user_options>
```

```
      <weight> default </weight>
      <tolc> default </tolc>
      <prob_bound>  default </prob_bound>
      <nstuck_solution> default  </nstuck_solution>
    </user_options>
    <global_options>
      <dim_ref> default </dim_ref>
      <ndiverse> default </ndiverse>
      <combination> default </combination>
      <n_stuck> default </n_stuck>
    </global_options>
    <local_options>
      <solver> misqp </solver>
      <tol> 2 </tol>
      <evalmax> 5000 </evalmax>
      <iterprint> 0 </iterprint>
      <n1> 1 </n1>
      <n2> 10 </n2>
      <balance> 0.25 </balance>
      <bestx> default </bestx>
    </local_options>
  </method>
  <parallelization name="cooperative">
    <reception_threshold> 1 </reception_threshold>
    <evals_threshold> 5000 </evals_threshold>
    <mult_num_sendSol> 10 </mult_num_sendSol>
    <minimum_num_sendSol> 20 </minimum_num_sendSol>
    <migration_max_time> 10 </migration_max_time>
  </parallelization>
  <problem>
    <dim> 4 </dim>
    <lb> 0d0,0,0,0 </lb>
    <ub> 10d0,10,10,10 </ub>
    <cu> 8,10,5 </cu>
    <cl> -inf,-inf,-inf </cl>
    <point> 3,4,5,1 </point>
  </problem>
</xml>
```

(4) In the input file above, the number of integer variables in the problem is set using:

```
<int_var> 3 </int_var>
```

(5) Also note that the number of constraints and their bounds are set using:

```
// Number of inequality constraints
<ineq> 3 </ineq>
// Lower bounds of nonlinear inequality constraints
<cu>  8,10,5 </cu>
// Upper bounds of nonlinear inequality constraints
<cl> -inf,-inf,-inf </cl>
```

(6) Compile the code and run like in the previous sections, e.g.:

```
$ make clean
$ make all
$ mpirun -np 4 bin/paralleltestbed inputs/TEMPLATE_MINLP_CONSTRAINTS_EXAMPLE.xml output/TEST
```

11.7 Use case: parameter estimation problems using nl2sol as local solver

The performance of saCeSS greatly depends on the use of efficient local solvers, which in turn should be selected based on the type of problem being solver. For parameter estimation problems, we recommend the use of nl2sol as local solver. However, in the case of noisy and/or very large parameter estimation problems, users might also want to test the performance of DHC. To apply nl2sol local solver to a customized problem:

(1) We need define in the objective function code two additional procedures: one to calculate the residual vector R and another to evaluate the Jacobian matrix. For illustrative purposes, check the toy example provided in:

examples/nl2sol_ example.c

```
#include <structure_parallel testbed.h>

void residuals(double *, double *, int *, void *);
void jacobian(double *, double *, int *, void *);

// FUNCTION: ROSENBROCK FUNCTION
// with DIMENSION=2 ---> FX = [X0, X1]
void* examplefunction(double *x, void *data) {
    experiment_total *exp1;
    output_function *res;
    double FX,FX2;
    int D;
    const char *error;

    exp1 = (experiment_total *) data;
    res = NULL;
    res = (output_function *) calloc(1,sizeof(output_function));
    D=(*exp1).test.bench.dim;

    if ( D != 2) {
        perror("Wrong dimension of the problem. This example was configured for D=2\n");
        exit(0);
    }
    FX = 100.0 * pow( x[1] - pow(x[0],2), 2.0)+pow(x[0]-1,2.0);
    res->value = FX;

    return res;
}

// FUNCTION FOR COMPUTING RESIDUAL VECTOR.
void residuals(double *x, double *R, int *nr, void *exp) {
    experiment_total *exp1;
    int D;
    exp1 = (experiment_total *) exp;
    D=(*exp1).test.bench.dim; // DIMENSION OF THE PROBLEM
```

```
    // Modify next code
    *nr=2;
    if (R == NULL) R = (double *) malloc(*nr*sizeof(double));
    R[0] = 10.0 * (x[1] - pow(x[0],2.0));
    R[1] = 1.0 - x[0];

}

// FUNCTION to EVALUATE THE JACOBIAN MATRIX.
void jacobian(double *x, double *J, int *nJ, void *exp) {
    experiment_total *exp1;
    int D;
    exp1 = (experiment_total *) exp;
    D=(*exp1).test.bench.dim; // DIMENSION OF THE PROBLEM

    // Modify next code
    *nJ = 4;
    if (J == NULL) J = (double *) malloc(*nJ * sizeof(double));
    J[0] = -20.0 * x[0]; // (0,0)
    J[2] = 10.0;         // (0,1)
    J[1] = -1.0;         // (1,0)
    J[3] = 0.0;          // (1,1)
}
```

(2) Follow the same steps as in section 9 adding the previous code as a customized optimization problem and defining its corresponding XML input file:

inputs/TEMPLATE_CUSTOM_NL2SOL.xml

```
<xml>
    <run>
        <typebench> customized </typebench>
        <id> 0 </id>
        <log_scale> 0 </log_scale>
        <output>  1 </output>
        <verbose> 1 </verbose>
        <local_search> 1 </local_search>
        <stopping_criteria>
                <maxevaluation> 1e10 </maxevaluation>
                <maxtime> 1e10 </maxtime>
                <vtr> 0.00000001 </vtr>
        </stopping_criteria>
    </run>

    <method name="saCeSS">
        <user_options>
            <weight> default </weight>
            <tolc> default </tolc>
```

```
            <prob_bound>  default </prob_bound>
            <nstuck_solution> default  </nstuck_solution>
        </user_options>
        <global_options>
            <dim_ref> default </dim_ref>
            <ndiverse> default </ndiverse>
            <combination> default </combination>
            <n_stuck> default </n_stuck>
        </global_options>
        <local_options>
            <solver> nl2sol </solver>
            <tol> 2 </tol>
            <evalmax> 5000 </evalmax>
            <iterprint> 0 </iterprint>
            <n1> 1 </n1>
            <n2> 10 </n2>
            <balance> 0.25 </balance>
            <bestx> default </bestx>
        </local_options>
    </method>

    <parallelization name="cooperative">
        <reception_threshold> 1 </reception_threshold>
        <evals_threshold> 1000000 </evals_threshold>
        <mult_num_sendSol> 10 </mult_num_sendSol>
        <minimum_num_sendSol> 20 </minimum_num_sendSol>
        <migration_max_time> 10 </migration_max_time>
    </parallelization>

    <problem>
        <dim> 2 </dim>
        <ineq> 0 </ineq>
        <neq> 0 </neq>
        <int_var> 0 </int_var>
        <bin_var> 0 </bin_var>
        <lb>
          -2.048d0,-2.048d0
        </lb>
        <ub>
         2.048d0,2.048d0
        </ub>
    </problem>
</xml>
```

(3) Compile the code and run like in the previous sections, e.g.:

```
$ make clean
$ make all
$ mpirun -np 2 bin/paralleltestbed inputs/TEMPLATE_CUSTOM_NL2SOL.xml output/TEST
```

```
==================================================
== PARALLEL SACESS OPTIMIZATION LIBRARY ==========
==================================================

************ GENERAL INFORMATION *******************

* SOLVER :: ScatterSearch
    Version solver : saCeSS - SELF-ADAPTED ASYNCHONOUS COOPERATIVE eSS
    Topology       : mixture between master-slave and star topology
    Max.Evals      : 10000000000.0
    Max.Time(s)    : 10000000000.0
    Local Solver   : nl2sol

* TOTAL PROCESSORS USED :: 2
    Number of MPI proc.                : 2
    Number of openMP threads per proc. : 1

* TYPE OF BENCKMARK :: customized
    Name                      : user benchmarks
    Number of parameters      : 2
    Value To Reach            : 0.0000000100
    [DEFAULT] Value To Reach : 0.0000000100

************ SLAVE INFORMATION *******************
WARNING: you are using nl2sol local solver. Remember you
need define the residuals in the objective function code.

ID SLAVE:   1        Refset size:   3      ndiverse:   20
[PROCESSOR ID=1] Initial Pop: NFunEvals: 25 Bestf:  0.4412723059D+00 CPUTime: 0.00

************ OPTIMIZATION BEGINS *****************

[SLAVE ID=1] fx = 0.0000000000 < VTR =0.0000000100 -- CURRENT TIME 0.001034 s
[MASTER] Best Known Solution fx = 0.0000000000 -- CURRENT TIME 0.001227

************ RESULTS *****************************

[PROGRAM FINISHED] Desired function value achieved.

ID   1  -- SEED RANDOM NUMBERS:         57657233627468.0
ID   0  -- SEED RANDOM NUMBERS:         28827141675996.0

BEST SOLUTION:
        1.00000000000000000000000000000
```

```
        1.000000000000000000000000000000
f(x)=0.000000000000000000000000000000

TOTAL TIME TO REACH VTR : 0.001034
TOTAL EVALUATIONS       : 55
AVERAGE ITERATIONS      : 1.000000
EVALUATION FAILED       : 0

************ GRAPHS GENERATED **********************

 GRAPH PATH               : output/TEST
 CONVERGENCE GRAPH MATLAB : output/TEST/convergence.m
 CONVERGENCE GRAPHs CSV   : output/TEST/convergence_id*.csv
 GANTT CHART MATLAB       : output/TEST/gantt_id0.m
 GANTT CHARTs CSV         : output/TEST/gantt_id*.csv
 PERCENTAGE GRAPH MATLAB  : output/TEST/percentage_id0.m
 PERCENTAGE GRAPH CSV     : output/TEST/percentage_id0.csv
```

11.8 Use case: working with clusters or supercomputers

The saCeSS library is specially appropriate to work in current multicore clusters or supercomputers. Working in these parallel machines requires a good training in specific technologies, such as environment modules, queue systems, and others. This section shows two different examples of how to execute saCeSS in a multicore cluster, intended to help inexpert users. However, these examples are only illustrative and many parameters can change depending of the underlying infrastructure and software.

Most current multicore clusters and supercomputers use the utility *modules* to manage almost all software, aiming to be able to support different versions of the same package/software library. This way, users can switch versions without having to explicitly specify different paths, besides that the environment variables are automatically managed. By default, most systems load some modules when a user logs in the system (you can see them using "module list"). Other modules should be manually loaded by the user using the command "module load modulefile". Take this into account to compile and run saCeSS in your cluster.

In most systems, the execution of jobs will be performed exclusively on the compute nodes using a job scheduler, which will be responsible to distribute the jobs and obtain the necessary resources for them. Therefore, users send their jobs to the job scheduler (`qsub`) and they will be executed on the compute nodes when the manager deems appropriate (depending on system load and number of users, number of total jobs, policies, etc). A parallel batch job obtains a reserve within a parallel environment (PE). PEs allow reserve the required resources, like slots (CPU cores) in one or more compute nodes for execution of the parallel job. Therefore, a parallel batch job must identify a PE to instantiate, usually using the parameter `-pe pe_name slots`, specifying the number of slots to reserve.

Most systems would have already predefined different PEs (with different `pe_name`) that vary the way of reserving the slots required in the compute nodes. For example, one predefined PE can be called `mpi_rr` and use a round robin allocation of slots, trying to maximize the number of nodes used. If it is instanced with two slots, it should provide two nodes with a reserved slot in each one. Another predefined PE could be called `mpi_fu` and use a fill up strategy, attempting minimize the number of nodes used reserving the largest number of slots per node. If it is instanced with two slots, it should provide a single node with two reserved slots. When submitting hybrid MPI+OpenMP jobs note that the number of slots specified to the parallel environment should be the total number of cores required for the parallel job, calculated as: number of MPI processes x number of OpenMP threads per process.

Example 1: MPI parallel job.

(1) Create a script to send one job to the queue. A typical script *job.sh* can be found in the folder examples:

```
$cat examples/job.sh

# You need load the required modules, in this case the intel
# compilers and intel mpi.

  module load intel/2.144
  module load impi/5.0.0.028

# It is very important to set the number of the OMP variables
```

```
# on the correspondent system variable. In this case the value
# have to be equal to 1 (only one thread per MPI processor).

  export OMP_NUM_THREADS=1

# This is the typical call to saCeSS solver with mpirun. Do not
# change the name of variable $NSLOTS, because it loads the number
# of procesors specified in qsub call. $1 and $2 are the input
# parameters of the script, corresponding with path of the XML
# and output files.

  mpirun -np  $NSLOTS $HOME/SACESS_TOOLS/bin/paralleltestbed  $1 $2
```

(2) Check the permissions of the script and change them if required.

(3) Submit the job using the command qsub and the right options. In this case the `pe_fu` environment was used to minimize the number of nodes used:

```
qsub -S /bin/bash -cwd -N TEST -pe pe_fu 10 job.sh \
       inputs/TEMPLATE_CIRCADIAN.xml output/TEST

# -cwd : run in current working directory.
#
# -N TEST : Name of the job.
#
# -pe pe_fu 10 : parallel environment and number of processors. There are
# different types of PE and each one describes how the parallel processors of
# one job (in this case 10 MPI processes) are going to be distributed in the
# nodes of the cluster. In this case, pe_full (full up) is a strategy where
# the slots of one node are occupied by the processors until full the node,
# and if they need more, the same strategy begins in other node.
# More information with the command "qconf -spl".
#
# inputs/TEMPLATE_CIRCADIAN.xml : input XML of saCeSS.
#
# output/TEST : output path of saCeSS.
```

(4) Check the status of the jobs with the command: qstat. If there are any problems during the execution, you will need to check the error file generated by the queue system.

Example 2: MPI and openMP hybrid parallel job.

(1) In this example, we will create a script to send one job to the queue with the following configuration: 11 mpi processor and 3 openMP threads for each MPI processor. Check the script *job_hybrid.sh* in the folder examples:

```
# You need load the required modules, in this case the intel
# compilers and intel mpi.
```

```
module load intel/2.144
module load impi/5.0.0.028

# It is very important to set the OMP variables
# on the correspondent system variable.

export OMP_NUM_THREADS=3

# This is the typical call to saCeSS solver with mpirun for hybrid jobs.
# Do not  use the variable $NSLOTS, because now the number of slots asked
# by qsub command (mpi+openmp) is different than the number of MPI processes.
# The parameter --map-by slot:pe=$OMP_NUM_THREADS means that for each mpi
# process, the job needs a number of slots equal to the $OMP_NUM_THREADS
# value. We need to set this parameter for right allocation of the jobs in
# the resources, avoiding the oversubscription.
# Check if hyperthreading is enabled in your cluster, because if that is the case
# you will probably need to map your jobs with the double of slots ($OMP_NUM_THREADS x 2).
# $1 and $2 are the input parameters of the script, corresponding
#  with path of the XML and output files.


mpirun  --map-by slot:pe=$OMP_NUM_THREADS -np  10 $HOME/SACESS_TOOLS/bin/paralleltestbed  $1 $2
```

(2) Check the permissions of the script and change them if required.

(3) Please note that the correct allocation of jobs in hybrid MPI/OpenMP parallel programming is critical: you need to know the number of slots of the nodes where the jobs are going to be executed and define a suitable PE. To run the hybrid MPI/OpenMP configuration we need to reserve for: number of MPI processes x number of OpenMP threads per process, in this example 3x3=9 slots. Since our cluster has 8 slots per node, we can not use the previous `pe_fu` environment, because we need to allocate the OpenMP threads in the same node as their associate MPI process. Thus, we need to allocate the slots in groups of three, and, in our cluster, this is achieved using the predefined environment called `mpi_6p`, that reserves six slots per node.

```
qsub -S /bin/bash -cwd -N TEST -pe mpi_6p 30 job_hybrid.sh \
                     inputs/TEMPLATE_CIRCADIAN.xml  output/TEST

# -cwd : run in current working directory.
#
# -N TEST : Name of the job.
#
# -pe mpi_6p 30: parallel environment and number of processors. In this
# case 30 slots is needed ( 10 MPI processes x 3 openMP threads).
# The number of save slot must to be multiple of PE rule of allocation,
# for this reason sometimes you need save more slots than you need.
# More information with the command "qconf -spl".
#
# inputs/TEMPLATE_CIRCADIAN.xml : input XML of saCeSS.
#
# output/TEST : output path of saCeSS.
```

(4) Check the status of the jobs with the command: qstat. If there are any problems during the execution, you will need to check the error file generated by the queue system

12 Copyright and License Information

In this section, information about licenses and copyrights are given, including those of third-party codes, such as the BLAS or HDF5 libraries.

12.1 saCeSS library license

12.2 saCeSS library documentation license.

12.3 Third party licenses

This subsection contains licensing information about third-party products included.

12.3.1 BLAS

The reference BLAS (version 3.6.0) is a freely-available software package. It is available from netlib via anonymous ftp. They only ask that proper credit be given to the authors. Like all software, it is copyrighted. It is not trademarked, but they do request the following:

- If you modify the source for these routines we ask that you change the name of the routine and comment the changes made to the original.
- We will gladly answer any questions regarding the software. If a modification is done, however, it is the responsibility of the person who modified the routine to provide support.

12.3.2 HDF5

HDF5 version 1.8.12 is a data model, library, and file format for storing and managing data. The copyright for that code is as follows:

```
Copyright Notice and License Terms for
HDF5 (Hierarchical Data Format 5) Software Library and Utilities
-----------------------------------------------------------------------------

HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 2006-2015 by The HDF Group.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
```

12.3.3 libxml2

Libxml2 (version 2.9.1) is the XML C parser and toolkit developed for the Gnome project (but usable outside of the Gnome platform), and it is free software available under the MIT License:

```
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FIT-
NESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

12.3.4 GSL

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License.

```
Copyright (C) 1996, 1997, 1998, 1999, 2000, 2007 Brian Gough

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
```

13 Acknowledgments

The development of the saCeSS optimization library has received financial support from the Spanish Ministerio de Economía y Competitividad (and the FEDER) through the project "MultiScales" (subprojects DPI2011-28112-C04-03, DPI2011-28112-C04-04), “SYNBIOFACTORY” (DPI2014-55276-C5-2-R), and TIN2013-42148-P, and from the Galician Government under the Consolidation Program of Competitive Research Units (Network ref. R2014/041 and competitive reference groups GRC2013/055). DRP acknowledges financial support from the MICINN-FPI programme.

References

[1] Rodriguez-Fernandez, M., Egea, J.A., Banga, J.R.: Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. BMC Bioinformatics **7:483** (2006)

[2] Egea, J.A., Martí, R., Banga, J.R.: An evolutionary method for complex-process optimization. Computers & Operations Research **37**(2), 315–324 (2010)

[3] Jia, G., Stephanopoulos, G., Gunawan, R.: Incremental parameter estimation of kinetic metabolic network models. BMC Systems Biology **6**(1), 1 (2012)

[4] Becker, K., Balsa-Canto, E., Cicin-Sain, D., Hoermann, A., Janssens, H., Banga, J.R., Jaeger, J.: Reverse-engineering post-transcriptional regulation of gap genes in *Drosophila melanogaster*. PLoS Comput Biol **9**(10), 1003281 (2013)

[5] de Hijas-Liste, G.M., Klipp, E., Balsa-Canto, E., Banga, J.R.: Global dynamic optimization approach to predict activation in metabolic pathways. BMC Systems Biology **8**(1), 1 (2014)

[6] Henriques, D., Rocha, M., Saez-Rodriguez, J., Banga, J.R.: Reverse engineering of logic-based differential equation models using a mixed-integer dynamic optimization approach. Bioinformatics **31**(18), 2999–3007 (2015)

[7] Gábor, A., Banga, J.R.: Robust and efficient parameter estimation in dynamic models of biological systems. BMC Systems B iology **9**(1), 74 (2015)

[8] Fan, M., Kuwahara, H., Wang, X., Wang, S., Gao, X.: Parameter estimation methods for gene circuit modeling from time-series mrna data: a comparative study. Briefings in bioinformatics **16**(6), 987–999 (2015)

[9] Villaverde, A.F., Egea, J.A., Banga, J.R.: A cooperative strategy for parameter estimation in large scale systems biology models. BMC Systems Biology **6:75** (2012)

[10] Penas, D.R., González, P., Egea, J.A., Banga, J.R., Doallo, R.: Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method. Procedia Computer Science **51**, 630–639 (2015). International Conference On Computational Science (ICCS 2015)

[11] Dennis, J.E. Jr., Gay, D.M., Welsch, R.E.: Algorithm 573: Nl2sol—an adaptive nonlinear least-squares algorithm [e4]. ACM Trans. Math. Softw. **7**(3), 369–383 (1981)

[12] de la Maza, M., Yuret, D.: Dynamic hill climbing. AI Expert **9**(3), 26–31 (1994)

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

[14] Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA (2009)

[15] Villaverde, A.F., Henriques, D., Smallbone, K., Bongard, S., Schmid, J., Cicin-Sain, D., Crombach, A., Saez-Rodriguez, J., Mauch, K., Balsa-Canto, E., Mendes, P., Jaeger, J., Banga, J.R.: Biopredyn-bench: a suite of benchmark problems for dynamic modelling in systems biology. BMC Systems Biology **9:1-15** (2015)

[16] Locke, J.C.W., Millar, A.J., Turner, M.S.: Modelling genetic networks with noisy and varied experimental data: The circadian clock in arabidopsis thaliana. Journal of Theoretical Biology **234**(3), 383–393 (2005)

[17] Moles, C.G., Mendes, P., R. Banga, J.: Parameter estimation in biochemical pathways: A comparison of global optimization methods. Genome Research **13**(11), 2467–2474 (2003)

[18] Lipniacki, T., Paszek, P., Brasier, A.R., Luxon, B., Kimmel, M.: Mathematical model of nf-κb regulatory module. Journal of Theoretical Biology **228**(2), 195–215 (2004)

[19] Egea, J.A., Henriques, D., Cokelaer, T., Villaverde, A.F., MacNamara, A., Danciu, D.-P., Banga, J.R., Saez-Rodriguez, J.: Meigo: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. BMC Bioinformatics **15**(1), 1–9 (2014)