MPCOTool

2.1.1

Generated by Doxygen 1.8.9.1

Fri Jan 22 2016 17:29:24

Contents

Chapter 1

MPCOTool

The Multi-Purposes Calibration and Optimization Tool. A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 2.0.1: Stable and recommended version.
- 2.1.2: Developing version to do new features.

AUTHORS

- Javier Burguete Tolosa (jburguete@eead.csic.es)
- Borja Latorre Garcés (borja.latorre@csic.es)

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

- gcc or clang (to compile the source code)
- make (to build the executable file)
- autoconf (to generate the Makefile in different operative systems)
- automake (to check the operative system)
- pkg-config (to find the libraries to compile)
- gsl (to generate random numbers)
- libxml (to deal with XML files)
- glib (extended utilities of C to work with data, lists, mapped files, regular expressions, using multicores in shared memory machines, ...)
- genetic (genetic algorithm)

OPTIONAL TOOLS AND LIBRARIES

- gettext (to work with different locales)
- gtk+ (to create the interactive GUI tool)
- openmpi or mpich (to run in parallelized tasks on multiple computers)

- `doxygen` (standard comments format to generate documentation)
- `latex` (to build the PDF manuals)

FILES

The source code has to have the following files:

- 2.0.1/configure.ac: configure generator.
- 2.0.1/Makefile.in: Makefile generator.
- 2.0.1/config.h.in: config header generator.
- 2.0.1/mpcotool.c: main source code.
- 2.0.1/mpcotool.h: main header code.
- 2.0.1/interface.h: interface header code.
- 2.0.1/build: script to build all.
- 2.0.1/logo.png: logo figure.
- 2.0.1/Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- README.md: this file.
- tests/testX/∗: several tests to check the program working.
- locales/∗/LC_MESSAGES/mpcotool.po: translation files.
- manuals/∗.eps: manual figures in EPS format.
- manuals/∗.png: manual figures in PNG format.
- manuals/∗.tex: documentation source files.
- applications/∗/∗: several practical application cases.
- check_errors/∗.xml: several mistaken files to check error handling.

BUILDING INSTRUCTIONS

This software has been built and tested in the following operative systems. Probably, it can be built in other systems, distributions, or versions but it has not been tested.

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

FreeBSD 10.2

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

1. Download the latest `genetic` doing on a terminal:

$ git clone https://github.com/jburguete/genetic.git

2. Download this repository:

   $ git clone https://github.com/jburguete/mpcotool.git

3. Link the latest genetic version to genetic:

   $ cd mpcotool/2.0.1
   $ ln -s ../../genetic/1.0.0 genetic

4. Build doing on a terminal:

   $ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

2. Then, in a terminal, follow steps 1 to 4 of the previous Debian 8 section.

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

1. Install MSYS2 and the required libraries and utilities. You can follow detailed instructions in install-unix
2. Then, in a MSYS2 terminal, follow steps 1 to 4 of the previous Debian 8 section.
3. Optional Windows binary package can be built doing in the terminal:

   $ make windist

Fedora Linux 23

1. In order to use OpenMPI compilation do in a terminal (in 64 bits version):

   $ export PATH=$PATH:/usr/lib64/openmpi/bin

2. Then, follow steps 1 to 4 of the previous Debian 8 section.

MAKING MANUALS INSTRUCTIONS

On UNIX type systems you need texlive installed. On Windows systems you need MiKTeX. In order to compile the manuals you can type on a terminal:

$ make manuals

MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory mpcotool/2.0.1):

```
$ cd ../tests/test2
$ ln -s ../../../genetic/1.0.0 genetic
$ cd ../test3
$ ln -s ../../../genetic/1.0.0 genetic
$ cd ../test4
$ ln -s ../../../genetic/1.0.0 genetic
```

2. Build all tests doing in the same terminal:

```
$ cd ../../2.0.1
$ make tests
```

USER INSTRUCTIONS

- Command line in sequential mode:

```
$ ./mpcotoolbin [-nthreads X] input_file.xml
```

- Command line in parallelized mode (where X is the number of threads to open in every node):

```
$ mpirun [MPI options] ./mpcotoolbin [-nthreads X] input_file.xml
```

- The syntax of the simulator has to be:

```
$ ./simulator_name input_file_1 [input_file_2] [input_file_3] [input_file_4] output_file
```

- The syntax of the program to evaluate the objetive function has to be (where the first data in the results file has to be the objective function value):

```
$ ./evaluator_name simulated_file data_file results_file
```

- On UNIX type systems the GUI application can be open doing on a terminal:

```
$ ./mpcotool
```

INPUT FILE FORMAT

The format of the main input file is as:

“‘xml <?xml version="1.0"?> <calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm↩
_type" nsimulations="simulations_number" niterations="iterations_number" tolerance="tolerance_value" nbest="best↩
_number" npopulation="population_number" ngenerations="generations_number" mutation="mutation_↩
ratio" reproduction="reproduction_ratio" adaptation="adaptation_ratio" gradient_type="gradient_method_type"
nsteps="steps_number" relaxation="relaxation_paramter" nestimates="estimates_number" seed="random_↩
seed" result="result_file" variables="variables_file"> <experiment name="data_file_1" template1="template_1_1"
template2="template_1_2" ... weight="weight_1"/> ... <experiment name="data_file_N" template1="template↩
_N_1" template2="template_N_2" ... weight="weight_N"/> <variable name="variable_1" minimum="min_value"
maximum="max_value" precision="precision_digits" sweeps="sweeps_number" nbits="bits_number" step="step↩
_size"> ... <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_↩
digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> </calibrate> “‘

with:

- **simulator**: simulator executable file name.
- **evaluator**: Optional. When needed is the evaluator executable file name.
- **seed**: Optional. Seed of the pseudo-random numbers generator (default value is 7007).
- **result**: Optional. It is the name of the optime result file (default name is "result").
- **variables**: Optional. It is the name of all simulated variables file (default name is "variables").

- **precision**: Optional, defined for each variable. Number of precision digits to evaluate the variable. 0 apply for integer numbers (default value is 14).
- **weight** Optional, defined for each experiment. Multiplies the objective value obtained for each experiment in the final objective function value (default value is 1).

Implemented algorithms are:

- **sweep**: Sweep brute force algorithm. It requires for each variable:
  - *sweeps*: number of sweeps to generate for each variable in every experiment.
    The total number of simulations to run is:
    (number of experiments) x (variable 1 number of sweeps) x ... x (variable n number of sweeps) x (number of iterations)
- **Monte-Carlo**: Monte-Carlo brute force algorithm. It requires on calibrate:
  - *nsimulations*: number of simulations to run in every experiment.
    The total number of simulations to run is:
    (number of experiments) x (number of simulations) x (number of iterations)
- Both brute force algorithms can be iterated to improve convergence by using the following parameters:
  - *nbest*: number of best simulations to calculate convergence interval on next iteration (default 1).
  - *tolerance*: tolerance parameter to increase convergence interval (default 0).
  - *niterations*: number of iterations (default 1).
    It multiplies the total number of simulations:
    x (number of iterations)
- Moreover, both brute force algorithms can be coupled with a gradient based method by using:
  - *gradient_type*: method to estimate the gradient. Two options are currently available:
    - coordinates: coordinates descent method.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x 2 x (number of variables)
    - random: random method. It requires:
    - nestimates: number of random checks to estimate the gradient.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x (number of estimates)

  Both methods require also:
  - nsteps: number of steps to perform the gradient based method,
  - relaxation: relaxation parameter,

  and for each variable:
  - step: initial step size for the gradient based method.
- **genetic**: Genetic algorithm. It requires the following parameters:
  - *npopulation*: number of population.
  - *ngenerations*: number of generations.
  - *mutation*: mutation ratio.
  - *reproduction*: reproduction ratio.
  - *adaptation*: adaptation ratio.

and for each variable:

- *nbits*: number of bits to encode each variable.

The total number of simulations to run is:

(number of experiments) x (npopulation) x [1 + (ngenerations - 1) x (mutation + reproduction + adaptation)]

SOME EXAMPLES OF INPUT FILES

Example 1

- The simulator program name is: *pivot*
- The syntax is:

  $ ./pivot input_file output_file

- The program to evaluate the objective function is: *compare*
- The syntax is:

  $ ./compare simulated_file data_file result_file

- The calibration is performed with a *sweep brute force algorithm*.
- The experimental data files are:

  27-48.txt
  42.txt
  52.txt
  100.txt

- Templates to get input files to simulator for each experiment are:

  template1.js
  template2.js
  template3.js
  template4.js

- The variables to calibrate, ranges, precision and sweeps number to perform are:

  alpha1, [179.70, 180.20], 2, 5
  alpha2, [179.30, 179.60], 2, 5
  random, [0.00, 0.20], 2, 5
  boot-time, [0.0, 3.0], 1, 5

- Then, the number of simulations to run is: 4x5x5x5x5=2500.
- The input file is:

“‘xml <?xml version="1.0"?> <calibrate simulator="pivot" evaluator="compare" algorithm="sweep"> <experiment name="27-48.txt" template1="template1.js"> <experiment name="42.txt" template1="template2.js"> <experiment name="52.txt" template1="template3.js"> <experiment name="100.txt" template1="template4.js"> <variable name="alpha1" minimum="179.70" maximum="180.20" precision="2" nsweeps="5"> <variable name="alpha2" minimum="179.30" maximum="179.60" precision="2" nsweeps="5"> <variable name="random" minimum="0.↩ 00" maximum="0.20" precision="2" nsweeps="5"> <variable name="boot-time" minimum="0.0" maximum="3.0" precision="1" nsweeps="5"> </calibrate> “‘

- A template file as *template1.js*:

“‘ { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.02824, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03008, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03753, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

- produces simulator input files to reproduce the experimental data file *27-48.txt* as:

“‘json { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.02824, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03008, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03753, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

- char * template [MAX_NINPUTS]

  Array of input template names.
- char * name

  File name.
- double weight

  Weight to calculate the objective function value.

4.1.1 Detailed Description

Struct to define experiment data.

Definition at line 48 of file interface.h.

The documentation for this struct was generated from the following file:

- interface.h

4.2 Input Struct Reference

Struct to define the optimization input file.

`#include <optimize.h>`

Data Fields

- char ** template [MAX_NINPUTS]

  Matrix of template names of input files.
- char ** experiment

  Array of experimental data file names.

- char ** label

  Array of variable names.
- char * result

  Name of the result file.
- char * variables

  Name of the variables file.
- char * simulator

  Name of the simulator program.
- char * evaluator

  Name of the program to evaluate the objective function.
- char * directory

  Working directory.
- char * name

  Input data file name.
- double * rangemin

  Array of minimum variable values.
- double * rangemax

  Array of maximum variable values.
- double * rangeminabs

  Array of absolute minimum variable values.
- double * rangemaxabs

  Array of absolute maximum variable values.
- double * weight

  Array of the experiment weights.
- double * step

  Array of direction search method step sizes.
- unsigned int * precision

  Array of variable precisions.
- unsigned int * nsweeps

  Array of sweeps of the sweep algorithm.
- unsigned int * nbits

  Array of bits numbers of the genetic algorithm.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- double p

  Exponent of the P error norm.
- double thresold

  Thresold to finish the optimization.
- unsigned long int seed

  Seed of the pseudo-random numbers generator.
- unsigned int nvariables

  Variables number.
- unsigned int nexperiments

Experiments number.

- unsigned int ninputs

  Number of input files to the simulator.
- unsigned int nsimulations

  Simulations number per experiment.
- unsigned int algorithm

  Algorithm type.
- unsigned int nsteps

  Number of steps to do the direction search method.
- unsigned int direction

  Method to estimate the direction search.
- unsigned int nestimates

  Number of simulations to estimate the direction search.
- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int norm

  Error norm type.

4.2.1 Detailed Description

Struct to define the optimization input file.

Definition at line 82 of file optimize.h.

The documentation for this struct was generated from the following file:

- optimize.h

4.3 Optimize Struct Reference

Struct to define the optimization ation data.

`#include <optimize.h>`

Data Fields

- GMappedFile ** file [MAX_NINPUTS]

  Matrix of input template files.
- char ** template [MAX_NINPUTS]

  Matrix of template names of input files.
- char ** experiment

  Array of experimental data file names.
- char ** label

  Array of variable names.
- gsl_rng * rng

  GSL random number generator.
- GeneticVariable * genetic_variable

  Array of variables for the genetic algorithm.
- FILE * file_result

Result file.

- FILE * file_variables

  Variables file.

- char * result

  Name of the result file.

- char * variables

  Name of the variables file.

- char * simulator

  Name of the simulator program.

- char * evaluator

  Name of the program to evaluate the objective function.

- double * value

  Array of variable values.

- double * rangemin

  Array of minimum variable values.

- double * rangemax

  Array of maximum variable values.

- double * rangeminabs

  Array of absolute minimum variable values.

- double * rangemaxabs

  Array of absolute maximum variable values.

- double * error_best

  Array of the best minimum errors.

- double * weight

  Array of the experiment weights.

- double * step

  Array of direction search method step sizes.

- double * direction

  Vector of direction search estimation.

- double * value_old

  Array of the best variable values on the previous step.

- double * error_old

  Array of the best minimum errors on the previous step.

- unsigned int * precision

  Array of variable precisions.

- unsigned int * nsweeps

  Array of sweeps of the sweep algorithm.

- unsigned int * thread

  Array of simulation numbers to calculate on the thread.

- unsigned int * thread_direction
- unsigned int * simulation_best

  Array of best simulation numbers.

- double tolerance

  Algorithm tolerance.

- double mutation_ratio

  Mutation probability.

- double reproduction_ratio

  Reproduction probability.

- double adaptation_ratio

  Adaptation probability.

- double relaxation

Relaxation parameter.

- double calculation_time

  Calculation time.

- double p

  Exponent of the P error norm.

- double thresold

  Thresold to finish the optimization.

- unsigned long int seed

  Seed of the pseudo-random numbers generator.

- unsigned int nvariables

  Variables number.

- unsigned int nexperiments

  Experiments number.

- unsigned int ninputs

  Number of input files to the simulator.

- unsigned int nsimulations

  Simulations number per experiment.

- unsigned int nsteps

  Number of steps for the direction search method.

- unsigned int nestimates

  Number of simulations to estimate the direction.

- unsigned int algorithm

  Algorithm type.

- unsigned int nstart

  Beginning simulation number of the task.

- unsigned int nend

  Ending simulation number of the task.

- unsigned int nstart_direction

  Beginning simulation number of the task for the direction search method.

- unsigned int nend_direction

  Ending simulation number of the task for the direction search method.

- unsigned int niterations

  Number of algorithm iterations.

- unsigned int nbest

  Number of best simulations.

- unsigned int nsaveds

  Number of saved simulations.

- unsigned int stop

  To stop the simulations.

- int mpi_rank

  Number of MPI task.

4.3.1 Detailed Description

Struct to define the optimization ation data.

Definition at line 133 of file optimize.h.

4.3.2 Field Documentation

4.3.2.1 unsigned int* Optimize::thread_direction

Array of simulation numbers to calculate on the thread for the direction search method.

Definition at line 167 of file optimize.h.

The documentation for this struct was generated from the following file:

- optimize.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_seed

  Pseudo-random numbers generator seed GtkLabel.
- GtkSpinButton * spin_seed

  Pseudo-random numbers generator seed GtkSpinButton.
- GtkLabel * label_threads

  Threads number GtkLabel.
- GtkSpinButton * spin_threads

  Threads number GtkSpinButton.
- GtkLabel * label_direction

  Direction threads number GtkLabel.
- GtkSpinButton * spin_direction

  Direction threads number GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 78 of file interface.h.

The documentation for this struct was generated from the following file:

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <optimize.h>`

Data Fields

- unsigned int thread
  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 209 of file optimize.h.

The documentation for this struct was generated from the following file:

- optimize.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog
  Main GtkDialog.
- GtkLabel * label
  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 96 of file interface.h.

The documentation for this struct was generated from the following file:

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label
  Variable label.
- double rangemin
  Minimum value.
- double rangemax
  Maximum value.
- double rangeminabs

Minimum allowed value.

- double rangemaxabs

  Maximum allowed value.

- double step

  Initial step size for the direction search method.

- unsigned int precision

  Precision digits.

- unsigned int nsweeps

  Sweeps number of the sweep algorithm.

- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 60 of file interface.h.

The documentation for this struct was generated from the following file:

- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:

Data Fields

- GtkWindow * window

  Main GtkWindow.

- GtkGrid * grid

  Main GtkGrid.

- GtkToolbar * bar_buttons

  GtkToolbar to store the main buttons.

- GtkToolButton * button_open

  Open GtkToolButton.

- GtkToolButton * button_save

  Save GtkToolButton.

- GtkToolButton * button_run

  Run GtkToolButton.

- GtkToolButton * button_options

  Options GtkToolButton.

- GtkToolButton * button_help

  Help GtkToolButton.

- GtkToolButton * button_about

  Help GtkToolButton.

- GtkToolButton * button_exit

  Exit GtkToolButton.

- GtkGrid * grid_files

  Files GtkGrid.
- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkLabel * label_result

  Result file GtkLabel.
- GtkEntry * entry_result

  Result file GtkEntry.
- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- GtkFrame * frame_norm

  GtkFrame to set the error norm.
- GtkGrid * grid_norm

  GtkGrid to set the error norm.
- GtkRadioButton * button_norm [NNORMS]

  Array of GtkButtons to set the error norm.
- GtkLabel * label_p

  GtkLabel to set the p parameter.
- GtkSpinButton * spin_p

  GtkSpinButton to set the p parameter.
- GtkScrolledWindow * scrolled_p

  GtkScrolledWindow to set the p parameter.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

  GtkGrid to set the algorithm.
- GtkRadioButton * button_algorithm [NALGORITHMS]

  Array of GtkButtons to set the algorithm.
- GtkLabel * label_simulations

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

GtkSpinButton to set the best number.

- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton * check_direction

  GtkCheckButton to check running the direction search method.
- GtkGrid * grid_direction

  GtkGrid to pack the direction search method widgets.
- GtkRadioButton * button_direction [NDIRECTIONS]

  GtkRadioButtons array to set the direction estimate method.
- GtkLabel * label_steps

  GtkLabel to set the steps number.
- GtkSpinButton * spin_steps

  GtkSpinButton to set the steps number.
- GtkLabel * label_estimates

  GtkLabel to set the estimates number.
- GtkSpinButton * spin_estimates

  GtkSpinButton to set the estimates number.
- GtkLabel * label_relaxation

  GtkLabel to set the relaxation parameter.
- GtkSpinButton * spin_relaxation

  GtkSpinButton to set the relaxation parameter.
- GtkLabel * label_thresold

  GtkLabel to set the thresold.
- GtkSpinButton * spin_thresold

  GtkSpinButton to set the thresold.
- GtkScrolledWindow * scrolled_thresold

  GtkScrolledWindow to set the thresold.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.

- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkLabel * label_step

  GtkLabel to set the step.
- GtkSpinButton * spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow * scrolled_step

  step GtkScrolledWindow.
- GtkFrame * frame_experiment

Experiment GtkFrame.

- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

  GtkFileChooserButton to set the experimental data file.
- GtkLabel * label_weight

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

  Array of GtkCheckButtons to set the input templates.
- GtkFileChooserButton * button_template [MAX_NINPUTS]

  Array of GtkFileChooserButtons to set the input templates.
- GdkPixbuf * logo

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.
- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

  Identifier of the entry_variable signal.
- gulong id_template [MAX_NINPUTS]

  Array of identifiers of the check_template signal.
- gulong id_input [MAX_NINPUTS]

  Array of identifiers of the button_template signal.
- unsigned int nexperiments

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 106 of file interface.h.

The documentation for this struct was generated from the following file:

- interface.h

Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

This graph shows which files directly or indirectly include this file:

5.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2016, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00039 #ifndef CONFIG__H
00040 #define CONFIG__H 1
00041
00042 // Array sizes
00043
00044 #define MAX_NINPUTS 8
00045 #define NALGORITHMS 3
00047 #define NDIRECTIONS 2
00048 #define NNORMS 4
00049 #define NPRECISIONS 15
00050
00051 // Default choices
00052
00053 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00054 #define DEFAULT_RANDOM_SEED 7007
00055 #define DEFAULT_RELAXATION 1.
```

```
00056
00057 // Interface labels
00058
00059 #define LOCALE_DIR "locales"
00060 #define PROGRAM_INTERFACE "mpcotool"
00061
00062 // XML labels
00063
00064 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00065 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00067 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00069 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00071 #define XML_OPTIMIZE (const xmlChar*)"optimize"
00073 #define XML_COORDINATES (const xmlChar*)"coordinates"
00075 #define XML_DIRECTION (const xmlChar*)"direction"
00077 #define XML_EUCLIDIAN (const xmlChar*)"euclidian"
00079 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00081 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00083 #define XML_GENETIC (const xmlChar*)"genetic"
00085 #define XML_MINIMUM (const xmlChar*)"minimum"
00086 #define XML_MAXIMUM (const xmlChar*)"maximum"
00087 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00088 #define XML_MUTATION (const xmlChar*)"mutation"
00090 #define XML_NAME (const xmlChar*)"name"
00091 #define XML_NBEST (const xmlChar*)"nbest"
00092 #define XML_NBITS (const xmlChar*)"nbits"
00093 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00094 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00096 #define XML_NITERATIONS (const xmlChar*)"niterations"
00098 #define XML_NORM (const xmlChar*)"norm"
00100 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00101 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00103 #define XML_NSTEPS (const xmlChar*)"nsteps"
00105 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00106 #define XML_P (const xmlChar*)"p"
00107 #define XML_PRECISION (const xmlChar*)"precision"
00108 #define XML_RANDOM (const xmlChar*)"random"
00110 #define XML_RELAXATION (const xmlChar*)"relaxation"
00111 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00113 #define XML_RESULT (const xmlChar*)"result"
00115 #define XML_SIMULATOR (const xmlChar*)"simulator"
00116 #define XML_SEED (const xmlChar*)"seed"
00118 #define XML_STEP (const xmlChar*)"step"
00119 #define XML_SWEEP (const xmlChar*)"sweep"
00120 #define XML_TAXICAB (const xmlChar*)"taxicab"
00121 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00122 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00124 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00126 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00128 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00130 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00132 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00134 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00136 #define XML_THRESOLD (const xmlChar*)"thresold"
00138 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00140 #define XML_VARIABLE (const xmlChar*)"variable"
00142 #define XML_VARIABLES (const xmlChar*)"variables"
00143 #define XML_WEIGHT (const xmlChar*)"weight"
00145
00146 #endif
```

5.3 interface.c File Reference

Source file to define the graphical interface functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <alloca.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"
#include "interface.h"
```

Include dependency graph for interface.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG 0

  Macro to debug.
- #define INPUT_FILE "test-ga.xml"

  Macro to define the initial input file.

Functions

- void input_save_direction (xmlNode *node)

  Function to save the direction search method data in a XML node.
- void input_save (char *filename)

  Function to save the input file.
- void options_new ()

  Function to open the options dialog.
- void running_new ()

  Function to open the running dialog.
- unsigned int window_get_algorithm ()

  Function to get the stochastic algorithm number.
- unsigned int window_get_direction ()

  Function to get the direction search method number.
- unsigned int window_get_norm ()

  Function to get the norm method number.
- void window_save_direction ()

  Function to save the direction search method data in the input file.
- int window_save ()

  Function to save the input file.
- void window_run ()

  Function to run a optimization.
- void window_help ()

  Function to show a help dialog.
- void window_about ()

Function to show an about dialog.

- void window_update_direction ()

  Function to update direction search method widgets view in the main window.

- void window_update ()

  Function to update the main window view.

- void window_set_algorithm ()

  Function to avoid memory errors changing the algorithm.

- void window_set_experiment ()

  Function to set the experiment data in the main window.

- void window_remove_experiment ()

  Function to remove an experiment in the main window.

- void window_add_experiment ()

  Function to add an experiment in the main window.

- void window_name_experiment ()

  Function to set the experiment name in the main window.

- void window_weight_experiment ()

  Function to update the experiment weight in the main window.

- void window_inputs_experiment ()

  Function to update the experiment input templates number in the main window.

- void window_template_experiment (void *data)

  Function to update the experiment i-th input template in the main window.

- void window_set_variable ()

  Function to set the variable data in the main window.

- void window_remove_variable ()

  Function to remove a variable in the main window.

- void window_add_variable ()

  Function to add a variable in the main window.

- void window_label_variable ()

  Function to set the variable label in the main window.

- void window_precision_variable ()

  Function to update the variable precision in the main window.

- void window_rangemin_variable ()

  Function to update the variable rangemin in the main window.

- void window_rangemax_variable ()

  Function to update the variable rangemax in the main window.

- void window_rangeminabs_variable ()

  Function to update the variable rangeminabs in the main window.

- void window_rangemaxabs_variable ()

  Function to update the variable rangemaxabs in the main window.

- void window_step_variable ()

  Function to update the variable step in the main window.

- void window_update_variable ()

  Function to update the variable data in the main window.

- int window_read (char *filename)

  Function to read the input data of a file.

- void window_open ()

  Function to open the input data.

- void window_new ()

  Function to open the main window.

Variables

- const char * logo [ ]

  Logo pixmap.
- Options options [1]

  Options struct to define the options dialog.
- Running running [1]

  Running struct to define the running dialog.
- Window window [1]

  Window struct to define the main interface window.

5.3.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file interface.c.

5.3.2 Function Documentation

5.3.2.1 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 201 of file interface.c.

```
00202 {
00203   unsigned int i, j;
00204   char *buffer;
00205   xmlDoc *doc;
00206   xmlNode *node, *child;
00207   GFile *file, *file2;
00208
00209 #if DEBUG
00210   fprintf (stderr, "input_save: start\n");
00211 #endif
00212
00213   // Getting the input file directory
00214   input->name = g_path_get_basename (filename);
00215   input->directory = g_path_get_dirname (filename);
00216   file = g_file_new_for_path (input->directory);
00217
00218   // Opening the input file
00219   doc = xmlNewDoc ((const xmlChar *) "1.0");
00220
00221   // Setting root XML node
00222   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00223   xmlDocSetRootElement (doc, node);
00224
00225   // Adding properties to the root XML node
00226   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00227     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00228   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
00229     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
```

```
      variables);
00230   file2 = g_file_new_for_path (input->simulator);
00231   buffer = g_file_get_relative_path (file, file2);
00232   g_object_unref (file2);
00233   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00234   g_free (buffer);
00235   if (input->evaluator)
00236     {
00237       file2 = g_file_new_for_path (input->evaluator);
00238       buffer = g_file_get_relative_path (file, file2);
00239       g_object_unref (file2);
00240       if (xmlStrlen ((xmlChar *) buffer))
00241         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00242       g_free (buffer);
00243     }
00244   if (input->seed != DEFAULT_RANDOM_SEED)
00245     xml_node_set_uint (node, XML_SEED, input->seed);
00246
00247   // Setting the algorithm
00248   buffer = (char *) g_malloc (64);
00249   switch (input->algorithm)
00250     {
00251     case ALGORITHM_MONTE_CARLO:
00252       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00253       snprintf (buffer, 64, "%u", input->nsimulations);
00254       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00255       snprintf (buffer, 64, "%u", input->niterations);
00256       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00257       snprintf (buffer, 64, "%.3lg", input->tolerance);
00258       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00259       snprintf (buffer, 64, "%u", input->nbest);
00260       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00261       input_save_direction (node);
00262       break;
00263     case ALGORITHM_SWEEP:
00264       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00265       snprintf (buffer, 64, "%u", input->niterations);
00266       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00267       snprintf (buffer, 64, "%.3lg", input->tolerance);
00268       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00269       snprintf (buffer, 64, "%u", input->nbest);
00270       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00271       input_save_direction (node);
00272       break;
00273     default:
00274       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00275       snprintf (buffer, 64, "%u", input->nsimulations);
00276       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00277       snprintf (buffer, 64, "%u", input->niterations);
00278       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00279       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00280       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00281       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00282       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00283       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00284       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00285       break;
00286     }
00287   g_free (buffer);
00288   if (input->thresold != 0.)
00289     xml_node_set_float (node, XML_THRESOLD, input->
      thresold);
00290
00291   // Setting the experimental data
00292   for (i = 0; i < input->nexperiments; ++i)
00293     {
00294       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00295       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
00296       if (input->weight[i] != 1.)
00297         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
00298       for (j = 0; j < input->ninputs; ++j)
00299         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
00300     }
00301
00302   // Setting the variables data
00303   for (i = 0; i < input->nvariables; ++i)
00304     {
00305       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00306       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
00307       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
00308       if (input->rangeminabs[i] != -G_MAXDOUBLE)
00309         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
00310       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
```

```
00311       if (input->rangemaxabs[i] != G_MAXDOUBLE)
00312         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
00313       if (input->precision[i] != DEFAULT_PRECISION)
00314         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
00315       if (input->algorithm == ALGORITHM_SWEEP)
00316         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
00317       else if (input->algorithm == ALGORITHM_GENETIC)
00318         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
00319       if (input->nsteps)
00320         xml_node_set_float (child, XML_STEP, input->
      step[i]);
00321     }
00322
00323   // Saving the error norm
00324   switch (input->norm)
00325     {
00326     case ERROR_NORM_MAXIMUM:
00327       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00328       break;
00329     case ERROR_NORM_P:
00330       xmlSetProp (node, XML_NORM, XML_P);
00331       xml_node_set_float (node, XML_P, input->p);
00332       break;
00333     case ERROR_NORM_TAXICAB:
00334       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00335     }
00336
00337   // Saving the XML file
00338   xmlSaveFormatFile (filename, doc, 1);
00339
00340   // Freeing memory
00341   xmlFreeDoc (doc);
00342
00343 #if DEBUG
00344   fprintf (stderr, "input_save: end\n");
00345 #endif
00346 }
```

Here is the call graph for this function:

5.3.2.2 void input_save_direction (xmlNode * *node*)

Function to save the direction search method data in a XML node.

Parameters

node	XML node.

Definition at line 169 of file interface.c.

```
00170 {
00171 #if DEBUG
00172   fprintf (stderr, "input_save_direction: start\n");
00173 #endif
00174   if (input->nsteps)
00175     {
00176       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
00177       if (input->relaxation != DEFAULT_RELAXATION)
00178         xml_node_set_float (node, XML_RELAXATION,
      input->relaxation);
00179       switch (input->direction)
00180         {
00181         case DIRECTION_METHOD_COORDINATES:
00182           xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00183           break;
00184         default:
00185           xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00186           xml_node_set_uint (node, XML_NESTIMATES,
      input->nestimates);
00187         }
00188     }
00189 #if DEBUG
00190   fprintf (stderr, "input_save_direction: end\n");
00191 #endif
00192 }
```

Here is the call graph for this function:

5.3.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 450 of file interface.c.

```
00451 {
00452   unsigned int i;
00453 #if DEBUG
00454   fprintf (stderr, "window_get_algorithm: start\n");
00455 #endif
00456   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00457 #if DEBUG
00458   fprintf (stderr, "window_get_algorithm: %u\n", i);
00459   fprintf (stderr, "window_get_algorithm: end\n");
00460 #endif
00461   return i;
00462 }
```

Here is the call graph for this function:

5.3.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 470 of file interface.c.

```
00471 {
00472   unsigned int i;
00473 #if DEBUG
00474   fprintf (stderr, "window_get_direction: start\n");
00475 #endif
00476   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00477 #if DEBUG
00478   fprintf (stderr, "window_get_direction: %u\n", i);
00479   fprintf (stderr, "window_get_direction: end\n");
00480 #endif
00481   return i;
00482 }
```

Here is the call graph for this function:

5.3.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 490 of file interface.c.

```
00491 {
00492   unsigned int i;
00493 #if DEBUG
00494   fprintf (stderr, "window_get_norm: start\n");
00495 #endif
00496   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00497 #if DEBUG
00498   fprintf (stderr, "window_get_norm: %u\n", i);
00499   fprintf (stderr, "window_get_norm: end\n");
00500 #endif
00501   return i;
00502 }
```

Here is the call graph for this function:

5.3.2.6 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 1597 of file interface.c.

```
01598 {
01599   unsigned int i;
01600   char *buffer;
01601 #if DEBUG
01602   fprintf (stderr, "window_read: start\n");
01603 #endif
01604
01605   // Reading new input file
01606   input_free ();
01607   if (!input_open (filename))
01608     return 0;
01609
01610   // Setting GTK+ widgets data
01611   gtk_entry_set_text (window->entry_result, input->result);
01612   gtk_entry_set_text (window->entry_variables, input->
      variables);
01613   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01614   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01615                                  (window->button_simulator), buffer);
01616   g_free (buffer);
01617   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01618                                 (size_t) input->evaluator);
01619   if (input->evaluator)
01620     {
01621       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01622       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01623                                      (window->button_evaluator), buffer);
01624       g_free (buffer);
01625     }
01626   gtk_toggle_button_set_active
01627     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01628   switch (input->algorithm)
01629     {
01630     case ALGORITHM_MONTE_CARLO:
01631       gtk_spin_button_set_value (window->spin_simulations,
01632                                  (gdouble) input->nsimulations);
01633     case ALGORITHM_SWEEP:
01634       gtk_spin_button_set_value (window->spin_iterations,
01635                                  (gdouble) input->niterations);
01636       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01637       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01638       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
```

```
01639                                   input->nsteps);
01640       if (input->nsteps)
01641         {
01642           gtk_toggle_button_set_active
01643             (GTK_TOGGLE_BUTTON (window->button_direction
01644                                 [input->direction]), TRUE);
01645           gtk_spin_button_set_value (window->spin_steps,
01646                                      (gdouble) input->nsteps);
01647           gtk_spin_button_set_value (window->spin_relaxation,
01648                                      (gdouble) input->relaxation);
01649           switch (input->direction)
01650             {
01651             case DIRECTION_METHOD_RANDOM:
01652               gtk_spin_button_set_value (window->spin_estimates,
01653                                          (gdouble) input->nestimates);
01654             }
01655         }
01656       break;
01657     default:
01658       gtk_spin_button_set_value (window->spin_population,
01659                                  (gdouble) input->nsimulations);
01660       gtk_spin_button_set_value (window->spin_generations,
01661                                  (gdouble) input->niterations);
01662       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01663       gtk_spin_button_set_value (window->spin_reproduction,
01664                                  input->reproduction_ratio);
01665       gtk_spin_button_set_value (window->spin_adaptation,
01666                                  input->adaptation_ratio);
01667     }
01668   gtk_toggle_button_set_active
01669     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01670   gtk_spin_button_set_value (window->spin_p, input->p);
01671   gtk_spin_button_set_value (window->spin_thresold, input->
      thresold);
01672   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01673   g_signal_handler_block (window->button_experiment,
01674                           window->id_experiment_name);
01675   gtk_combo_box_text_remove_all (window->combo_experiment);
01676   for (i = 0; i < input->nexperiments; ++i)
01677     gtk_combo_box_text_append_text (window->combo_experiment,
01678                                     input->experiment[i]);
01679   g_signal_handler_unblock
01680     (window->button_experiment, window->
      id_experiment_name);
01681   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01682   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01683   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01684   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01685   gtk_combo_box_text_remove_all (window->combo_variable);
01686   for (i = 0; i < input->nvariables; ++i)
01687     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
01688   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01689   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01690   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01691   window_set_variable ();
01692   window_update ();
01693
01694 #if DEBUG
01695   fprintf (stderr, "window_read: end\n");
01696 #endif
01697   return 1;
01698 }
```

Here is the call graph for this function:

5.3.2.7 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 543 of file interface.c.

```
00544 {
00545   GtkFileChooserDialog *dlg;
00546   GtkFileFilter *filter;
00547   char *buffer;
00548
00549 #if DEBUG
00550   fprintf (stderr, "window_save: start\n");
00551 #endif
00552
00553   // Opening the saving dialog
00554   dlg = (GtkFileChooserDialog *)
00555     gtk_file_chooser_dialog_new (gettext ("Save file"),
00556                                  window->window,
00557                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00558                                  gettext ("_Cancel"),
00559                                  GTK_RESPONSE_CANCEL,
00560                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00561   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00562   buffer = g_build_filename (input->directory, input->name, NULL);
00563   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00564   g_free (buffer);
00565
00566   // Adding XML filter
00567   filter = (GtkFileFilter *) gtk_file_filter_new ();
00568   gtk_file_filter_set_name (filter, "XML");
00569   gtk_file_filter_add_pattern (filter, "*.xml");
00570   gtk_file_filter_add_pattern (filter, "*.XML");
00571   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00572
00573   // If OK response then saving
00574   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00575     {
00576
00577       // Adding properties to the root XML node
00578       input->simulator = gtk_file_chooser_get_filename
00579         (GTK_FILE_CHOOSER (window->button_simulator));
00580       if (gtk_toggle_button_get_active
00581           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00582         input->evaluator = gtk_file_chooser_get_filename
00583           (GTK_FILE_CHOOSER (window->button_evaluator));
00584       else
00585         input->evaluator = NULL;
00586       input->result
00587         = (char *) xmlStrdup ((const xmlChar *)
00588                               gtk_entry_get_text (window->entry_result));
00589       input->variables
00590         = (char *) xmlStrdup ((const xmlChar *)
00591                               gtk_entry_get_text (window->entry_variables));
00592
00593       // Setting the algorithm
00594       switch (window_get_algorithm ())
00595         {
00596         case ALGORITHM_MONTE_CARLO:
00597           input->algorithm = ALGORITHM_MONTE_CARLO;
00598           input->nsimulations
00599             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00600           input->niterations
00601             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00602           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00603           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00604           window_save_direction ();
00605           break;
00606         case ALGORITHM_SWEEP:
00607           input->algorithm = ALGORITHM_SWEEP;
00608           input->niterations
00609             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00610           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00611           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00612           window_save_direction ();
00613           break;
00614         default:
00615           input->algorithm = ALGORITHM_GENETIC;
00616           input->nsimulations
00617             = gtk_spin_button_get_value_as_int (window->spin_population);
00618           input->niterations
```

```
00619             = gtk_spin_button_get_value_as_int (window->spin_generations);
00620           input->mutation_ratio
00621             = gtk_spin_button_get_value (window->spin_mutation);
00622           input->reproduction_ratio
00623             = gtk_spin_button_get_value (window->spin_reproduction);
00624           input->adaptation_ratio
00625             = gtk_spin_button_get_value (window->spin_adaptation);
00626           break;
00627         }
00628       input->norm = window_get_norm ();
00629       input->p = gtk_spin_button_get_value (window->spin_p);
00630       input->thresold = gtk_spin_button_get_value (window->
      spin_thresold);
00631
00632       // Saving the XML file
00633       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00634       input_save (buffer);
00635
00636       // Closing and freeing memory
00637       g_free (buffer);
00638       gtk_widget_destroy (GTK_WIDGET (dlg));
00639 #if DEBUG
00640       fprintf (stderr, "window_save: end\n");
00641 #endif
00642       return 1;
00643     }
00644
00645   // Closing and freeing memory
00646   gtk_widget_destroy (GTK_WIDGET (dlg));
00647 #if DEBUG
00648   fprintf (stderr, "window_save: end\n");
00649 #endif
00650   return 0;
00651 }
```

Here is the call graph for this function:

5.3.2.8 void window_template_experiment (void * *data*)

Function to update the experiment i-th input template in the main window.

Parameters

data	Callback data (i-th input template).

Definition at line 1201 of file interface.c.

```
01202 {
01203   unsigned int i, j;
01204   char *buffer;
01205   GFile *file1, *file2;
01206 #if DEBUG
01207   fprintf (stderr, "window_template_experiment: start\n");
01208 #endif
01209   i = (size_t) data;
01210   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01211   file1
01212     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01213   file2 = g_file_new_for_path (input->directory);
01214   buffer = g_file_get_relative_path (file2, file1);
01215   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
01216   g_free (buffer);
01217   g_object_unref (file2);
01218   g_object_unref (file1);
01219 #if DEBUG
01220   fprintf (stderr, "window_template_experiment: end\n");
01221 #endif
01222 }
```

5.4 interface.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
```

```
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elif !defined (BSD)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #include "genetic/genetic.h"
00060 #include "utils.h"
00061 #include "optimize.h"
00062 #include "interface.h"
00063
00064 #define DEBUG 0
00065
00066
00070 #ifdef G_OS_WIN32
00071 #define INPUT_FILE "test-ga-win.xml"
00072 #else
00073 #define INPUT_FILE "test-ga.xml"
00074 #endif
00075
00076 const char *logo[] = {
00077   "32 32 3 1",
00078   "      c None",
00079   ".     c #0000FF",
00080   "+     c #FF0000",
00081   "                                ",
00082   "                                ",
00083   "                                ",
00084   "      .       .       .       ",
00085   "      .       .       .       ",
00086   "      .       .       .       ",
00087   "      .       .       .       ",
00088   "      .       .     +++       ",
00089   "      .       .    +++++      ",
00090   "      .       .    +++++      ",
00091   "      .       .    +++++      ",
00092   "    +++       .     +++     +++",
00093   "   +++++      .       .    +++++",
00094   "   +++++      .       .    +++++",
00095   "   +++++      .       .    +++++",
00096   "    +++       .       .     +++",
00097   "      .       .       .       ",
00098   "      .       .     +++       ",
00099   "      .       .    +++++      ",
00100   "      .       .    +++++      ",
00101   "      .       .    +++++      ",
00102   "      .       .     +++       ",
```

```
00103   "     .      .      .      .     ",
00104   "     .      .      .      .     ",
00105   "     .      .      .      .     ",
00106   "     .      .      .      .     ",
00107   "     .      .      .      .     ",
00108   "     .      .      .      .     ",
00109   "     .      .      .      .     ",
00110   "                                ",
00111   "                                ",
00112   "                                "
00113 };
00114
00115 /*
00116 const char * logo[] = {
00117 "32 32 3 1",
00118 "     c #FFFFFFFFFFFF",
00119 ".    c #00000000FFFF",
00120 "X    c #FFFF00000000",
00121 "                                ",
00122 "                                ",
00123 "                                ",
00124 "     .      .      .      .     ",
00125 "     .      .      .      .     ",
00126 "     .      .      .      .     ",
00127 "     .      .      .      .     ",
00128 "     .      .     XXX     .     ",
00129 "     .      .    XXXXX    .     ",
00130 "     .      .    XXXXX    .     ",
00131 "     .      .    XXXXX    .     ",
00132 "    XXX     .     XXX    XXX    ",
00133 "   XXXXX    .      .    XXXXX   ",
00134 "   XXXXX    .      .    XXXXX   ",
00135 "   XXXXX    .      .    XXXXX   ",
00136 "    XXX     .      .     XXX    ",
00137 "     .      .      .      .     ",
00138 "     .     XXX     .      .     ",
00139 "     .    XXXXX    .      .     ",
00140 "     .    XXXXX    .      .     ",
00141 "     .    XXXXX    .      .     ",
00142 "     .     XXX     .      .     ",
00143 "     .      .      .      .     ",
00144 "     .      .      .      .     ",
00145 "     .      .      .      .     ",
00146 "     .      .      .      .     ",
00147 "     .      .      .      .     ",
00148 "     .      .      .      .     ",
00149 "     .      .      .      .     ",
00150 "                                ",
00151 "                                ",
00152 "                                "};
00153 */
00154
00155 Options options[1];
00157 Running running[1];
00159 Window window[1];
00161
00168 void
00169 input_save_direction (xmlNode * node)
00170 {
00171 #if DEBUG
00172   fprintf (stderr, "input_save_direction: start\n");
00173 #endif
00174   if (input->nsteps)
00175     {
00176       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
00177       if (input->relaxation != DEFAULT_RELAXATION)
00178         xml_node_set_float (node, XML_RELAXATION,
      input->relaxation);
00179       switch (input->direction)
00180         {
00181         case DIRECTION_METHOD_COORDINATES:
00182           xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00183           break;
00184         default:
00185           xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00186           xml_node_set_uint (node, XML_NESTIMATES,
      input->nestimates);
00187         }
00188     }
00189 #if DEBUG
00190   fprintf (stderr, "input_save_direction: end\n");
00191 #endif
00192 }
00193
00200 void
00201 input_save (char *filename)
```

```
00202 {
00203   unsigned int i, j;
00204   char *buffer;
00205   xmlDoc *doc;
00206   xmlNode *node, *child;
00207   GFile *file, *file2;
00208
00209 #if DEBUG
00210   fprintf (stderr, "input_save: start\n");
00211 #endif
00212
00213   // Getting the input file directory
00214   input->name = g_path_get_basename (filename);
00215   input->directory = g_path_get_dirname (filename);
00216   file = g_file_new_for_path (input->directory);
00217
00218   // Opening the input file
00219   doc = xmlNewDoc ((const xmlChar *) "1.0");
00220
00221   // Setting root XML node
00222   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00223   xmlDocSetRootElement (doc, node);
00224
00225   // Adding properties to the root XML node
00226   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00227     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00228   if (xmlStrcmp ((const xmlChar *) input->variables,
    variables_name))
00229     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
    variables);
00230   file2 = g_file_new_for_path (input->simulator);
00231   buffer = g_file_get_relative_path (file, file2);
00232   g_object_unref (file2);
00233   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00234   g_free (buffer);
00235   if (input->evaluator)
00236     {
00237       file2 = g_file_new_for_path (input->evaluator);
00238       buffer = g_file_get_relative_path (file, file2);
00239       g_object_unref (file2);
00240       if (xmlStrlen ((xmlChar *) buffer))
00241         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00242       g_free (buffer);
00243     }
00244   if (input->seed != DEFAULT_RANDOM_SEED)
00245     xml_node_set_uint (node, XML_SEED, input->seed);
00246
00247   // Setting the algorithm
00248   buffer = (char *) g_malloc (64);
00249   switch (input->algorithm)
00250     {
00251     case ALGORITHM_MONTE_CARLO:
00252       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00253       snprintf (buffer, 64, "%u", input->nsimulations);
00254       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00255       snprintf (buffer, 64, "%u", input->niterations);
00256       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00257       snprintf (buffer, 64, "%.3lg", input->tolerance);
00258       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00259       snprintf (buffer, 64, "%u", input->nbest);
00260       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00261       input_save_direction (node);
00262       break;
00263     case ALGORITHM_SWEEP:
00264       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00265       snprintf (buffer, 64, "%u", input->niterations);
00266       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00267       snprintf (buffer, 64, "%.3lg", input->tolerance);
00268       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00269       snprintf (buffer, 64, "%u", input->nbest);
00270       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00271       input_save_direction (node);
00272       break;
00273     default:
00274       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00275       snprintf (buffer, 64, "%u", input->nsimulations);
00276       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00277       snprintf (buffer, 64, "%u", input->niterations);
00278       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00279       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00280       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00281       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00282       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00283       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00284       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00285       break;
00286     }
```

```
00287   g_free (buffer);
00288   if (input->thresold != 0.)
00289     xml_node_set_float (node, XML_THRESOLD, input->
      thresold);
00290
00291   // Setting the experimental data
00292   for (i = 0; i < input->nexperiments; ++i)
00293     {
00294       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00295       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
00296       if (input->weight[i] != 1.)
00297         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
00298       for (j = 0; j < input->ninputs; ++j)
00299         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
00300     }
00301
00302   // Setting the variables data
00303   for (i = 0; i < input->nvariables; ++i)
00304     {
00305       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00306       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
00307       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
00308       if (input->rangeminabs[i] != -G_MAXDOUBLE)
00309         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
00310       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
00311       if (input->rangemaxabs[i] != G_MAXDOUBLE)
00312         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
00313       if (input->precision[i] != DEFAULT_PRECISION)
00314         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
00315       if (input->algorithm == ALGORITHM_SWEEP)
00316         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
00317       else if (input->algorithm == ALGORITHM_GENETIC)
00318         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
00319       if (input->nsteps)
00320         xml_node_set_float (child, XML_STEP, input->
      step[i]);
00321     }
00322
00323   // Saving the error norm
00324   switch (input->norm)
00325     {
00326     case ERROR_NORM_MAXIMUM:
00327       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00328       break;
00329     case ERROR_NORM_P:
00330       xmlSetProp (node, XML_NORM, XML_P);
00331       xml_node_set_float (node, XML_P, input->p);
00332       break;
00333     case ERROR_NORM_TAXICAB:
00334       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00335     }
00336
00337   // Saving the XML file
00338   xmlSaveFormatFile (filename, doc, 1);
00339
00340   // Freeing memory
00341   xmlFreeDoc (doc);
00342
00343 #if DEBUG
00344   fprintf (stderr, "input_save: end\n");
00345 #endif
00346 }
00347
00352 void
00353 options_new ()
00354 {
00355 #if DEBUG
00356   fprintf (stderr, "options_new: start\n");
00357 #endif
00358   options->label_seed = (GtkLabel *)
00359     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00360   options->spin_seed = (GtkSpinButton *)
00361     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00362   gtk_widget_set_tooltip_text
00363     (GTK_WIDGET (options->spin_seed),
00364      gettext ("Seed to init the pseudo-random numbers generator"));
00365   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00366   options->label_threads = (GtkLabel *)
```

```
00367     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00368   options->spin_threads
00369     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00370   gtk_widget_set_tooltip_text
00371     (GTK_WIDGET (options->spin_threads),
00372      gettext ("Number of threads to perform the calibration/optimization for "
00373               "the stochastic algorithm"));
00374   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00375   options->label_direction = (GtkLabel *)
00376     gtk_label_new (gettext ("Threads number for the direction search method"));
00377   options->spin_direction
00378     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00379   gtk_widget_set_tooltip_text
00380     (GTK_WIDGET (options->spin_direction),
00381      gettext ("Number of threads to perform the calibration/optimization for "
00382               "the direction search method"));
00383   gtk_spin_button_set_value (options->spin_direction,
00384                              (gdouble) nthreads_direction);
00385   options->grid = (GtkGrid *) gtk_grid_new ();
00386   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00387   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00388   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00389                    0, 1, 1, 1);
00390   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00391                    1, 1, 1, 1);
00392   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00393                    0, 2, 1, 1);
00394   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00395                    1, 2, 1, 1);
00396   gtk_widget_show_all (GTK_WIDGET (options->grid));
00397   options->dialog = (GtkDialog *)
00398     gtk_dialog_new_with_buttons (gettext ("Options"),
00399                                  window->window,
00400                                  GTK_DIALOG_MODAL,
00401                                  gettext ("_OK"), GTK_RESPONSE_OK,
00402                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
00403                                  NULL);
00404   gtk_container_add
00405     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00406      GTK_WIDGET (options->grid));
00407   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00408     {
00409       input->seed
00410         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00411       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00412       nthreads_direction
00413         = gtk_spin_button_get_value_as_int (options->spin_direction);
00414     }
00415   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00416 #if DEBUG
00417   fprintf (stderr, "options_new: end\n");
00418 #endif
00419 }
00420
00425 void
00426 running_new ()
00427 {
00428 #if DEBUG
00429   fprintf (stderr, "running_new: start\n");
00430 #endif
00431   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00432   running->dialog = (GtkDialog *)
00433     gtk_dialog_new_with_buttons (gettext ("Calculating"),
00434                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00435   gtk_container_add
00436     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00437      GTK_WIDGET (running->label));
00438   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00439 #if DEBUG
00440   fprintf (stderr, "running_new: end\n");
00441 #endif
00442 }
00443
00449 unsigned int
00450 window_get_algorithm ()
00451 {
00452   unsigned int i;
00453 #if DEBUG
00454   fprintf (stderr, "window_get_algorithm: start\n");
00455 #endif
00456   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00457 #if DEBUG
00458   fprintf (stderr, "window_get_algorithm: %u\n", i);
00459   fprintf (stderr, "window_get_algorithm: end\n");
00460 #endif
```

```
00461   return i;
00462 }
00463
00469 unsigned int
00470 window_get_direction ()
00471 {
00472   unsigned int i;
00473 #if DEBUG
00474   fprintf (stderr, "window_get_direction: start\n");
00475 #endif
00476   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00477 #if DEBUG
00478   fprintf (stderr, "window_get_direction: %u\n", i);
00479   fprintf (stderr, "window_get_direction: end\n");
00480 #endif
00481   return i;
00482 }
00483
00489 unsigned int
00490 window_get_norm ()
00491 {
00492   unsigned int i;
00493 #if DEBUG
00494   fprintf (stderr, "window_get_norm: start\n");
00495 #endif
00496   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00497 #if DEBUG
00498   fprintf (stderr, "window_get_norm: %u\n", i);
00499   fprintf (stderr, "window_get_norm: end\n");
00500 #endif
00501   return i;
00502 }
00503
00508 void
00509 window_save_direction ()
00510 {
00511 #if DEBUG
00512   fprintf (stderr, "window_save_direction: start\n");
00513 #endif
00514   if (gtk_toggle_button_get_active
00515       (GTK_TOGGLE_BUTTON (window->check_direction)))
00516     {
00517       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00518       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00519       switch (window_get_direction ())
00520         {
00521         case DIRECTION_METHOD_COORDINATES:
00522           input->direction = DIRECTION_METHOD_COORDINATES;
00523           break;
00524         default:
00525           input->direction = DIRECTION_METHOD_RANDOM;
00526           input->nestimates
00527             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00528         }
00529     }
00530   else
00531     input->nsteps = 0;
00532 #if DEBUG
00533   fprintf (stderr, "window_save_direction: end\n");
00534 #endif
00535 }
00536
00542 int
00543 window_save ()
00544 {
00545   GtkFileChooserDialog *dlg;
00546   GtkFileFilter *filter;
00547   char *buffer;
00548
00549 #if DEBUG
00550   fprintf (stderr, "window_save: start\n");
00551 #endif
00552
00553   // Opening the saving dialog
00554   dlg = (GtkFileChooserDialog *)
00555     gtk_file_chooser_dialog_new (gettext ("Save file"),
00556                                  window->window,
00557                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00558                                  gettext ("_Cancel"),
00559                                  GTK_RESPONSE_CANCEL,
00560                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00561   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00562   buffer = g_build_filename (input->directory, input->name, NULL);
```

```
00563   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00564   g_free (buffer);
00565
00566   // Adding XML filter
00567   filter = (GtkFileFilter *) gtk_file_filter_new ();
00568   gtk_file_filter_set_name (filter, "XML");
00569   gtk_file_filter_add_pattern (filter, "*.xml");
00570   gtk_file_filter_add_pattern (filter, "*.XML");
00571   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00572
00573   // If OK response then saving
00574   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00575     {
00576
00577       // Adding properties to the root XML node
00578       input->simulator = gtk_file_chooser_get_filename
00579         (GTK_FILE_CHOOSER (window->button_simulator));
00580       if (gtk_toggle_button_get_active
00581           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00582         input->evaluator = gtk_file_chooser_get_filename
00583           (GTK_FILE_CHOOSER (window->button_evaluator));
00584       else
00585         input->evaluator = NULL;
00586       input->result
00587         = (char *) xmlStrdup ((const xmlChar *)
00588                               gtk_entry_get_text (window->entry_result));
00589       input->variables
00590         = (char *) xmlStrdup ((const xmlChar *)
00591                               gtk_entry_get_text (window->entry_variables));
00592
00593       // Setting the algorithm
00594       switch (window_get_algorithm ())
00595         {
00596         case ALGORITHM_MONTE_CARLO:
00597           input->algorithm = ALGORITHM_MONTE_CARLO;
00598           input->nsimulations
00599             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00600           input->niterations
00601             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00602           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00603           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00604           window_save_direction ();
00605           break;
00606         case ALGORITHM_SWEEP:
00607           input->algorithm = ALGORITHM_SWEEP;
00608           input->niterations
00609             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00610           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00611           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00612           window_save_direction ();
00613           break;
00614         default:
00615           input->algorithm = ALGORITHM_GENETIC;
00616           input->nsimulations
00617             = gtk_spin_button_get_value_as_int (window->spin_population);
00618           input->niterations
00619             = gtk_spin_button_get_value_as_int (window->spin_generations);
00620           input->mutation_ratio
00621             = gtk_spin_button_get_value (window->spin_mutation);
00622           input->reproduction_ratio
00623             = gtk_spin_button_get_value (window->spin_reproduction);
00624           input->adaptation_ratio
00625             = gtk_spin_button_get_value (window->spin_adaptation);
00626           break;
00627         }
00628       input->norm = window_get_norm ();
00629       input->p = gtk_spin_button_get_value (window->spin_p);
00630       input->thresold = gtk_spin_button_get_value (window->
      spin_thresold);
00631
00632       // Saving the XML file
00633       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00634       input_save (buffer);
00635
00636       // Closing and freeing memory
00637       g_free (buffer);
00638       gtk_widget_destroy (GTK_WIDGET (dlg));
00639 #if DEBUG
00640       fprintf (stderr, "window_save: end\n");
00641 #endif
00642       return 1;
00643     }
00644
```

```
00645   // Closing and freeing memory
00646   gtk_widget_destroy (GTK_WIDGET (dlg));
00647 #if DEBUG
00648   fprintf (stderr, "window_save: end\n");
00649 #endif
00650   return 0;
00651 }
00652
00657 void
00658 window_run ()
00659 {
00660   unsigned int i;
00661   char *msg, *msg2, buffer[64], buffer2[64];
00662 #if DEBUG
00663   fprintf (stderr, "window_run: start\n");
00664 #endif
00665   if (!window_save ())
00666     {
00667 #if DEBUG
00668       fprintf (stderr, "window_run: end\n");
00669 #endif
00670       return;
00671     }
00672   running_new ();
00673   while (gtk_events_pending ())
00674     gtk_main_iteration ();
00675   optimize_open ();
00676   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00677   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00678   msg2 = g_strdup (buffer);
00679   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00680     {
00681       snprintf (buffer, 64, "%s = %s\n",
00682                 optimize->label[i], format[optimize->
      precision[i]]);
00683       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00684       msg = g_strconcat (msg2, buffer2, NULL);
00685       g_free (msg2);
00686     }
00687   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
00688             optimize->calculation_time);
00689   msg = g_strconcat (msg2, buffer, NULL);
00690   g_free (msg2);
00691   show_message (gettext ("Best result"), msg, INFO_TYPE);
00692   g_free (msg);
00693   optimize_free ();
00694 #if DEBUG
00695   fprintf (stderr, "window_run: end\n");
00696 #endif
00697 }
00698
00703 void
00704 window_help ()
00705 {
00706   char *buffer, *buffer2;
00707 #if DEBUG
00708   fprintf (stderr, "window_help: start\n");
00709 #endif
00710   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
00711                               gettext ("user-manual.pdf"), NULL);
00712   buffer = g_filename_to_uri (buffer2, NULL, NULL);
00713   g_free (buffer2);
00714   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
00715 #if DEBUG
00716   fprintf (stderr, "window_help: uri=%s\n", buffer);
00717 #endif
00718   g_free (buffer);
00719 #if DEBUG
00720   fprintf (stderr, "window_help: end\n");
00721 #endif
00722 }
00723
00728 void
00729 window_about ()
00730 {
00731   static const gchar *authors[] = {
00732     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00733     "Borja Latorre Garcés <borja.latorre@csic.es>",
00734     NULL
00735   };
00736 #if DEBUG
00737   fprintf (stderr, "window_about: start\n");
00738 #endif
00739   gtk_show_about_dialog
00740     (window->window,
00741      "program_name", "MPCOTool",
00742      "comments",
```

```
00743       gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
00744                "A software to perform calibrations or optimizations of "
00745                "empirical parameters"),
00746       "authors", authors,
00747       "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00748       "version", "2.0.1",
00749       "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
00750       "logo", window->logo,
00751       "website", "https://github.com/jburguete/mpcotool",
00752       "license-type", GTK_LICENSE_BSD, NULL);
00753 #if DEBUG
00754   fprintf (stderr, "window_about: end\n");
00755 #endif
00756 }
00757
00763 void
00764 window_update_direction ()
00765 {
00766 #if DEBUG
00767   fprintf (stderr, "window_update_direction: start\n");
00768 #endif
00769   gtk_widget_show (GTK_WIDGET (window->check_direction));
00770   if (gtk_toggle_button_get_active
00771       (GTK_TOGGLE_BUTTON (window->check_direction)))
00772     {
00773       gtk_widget_show (GTK_WIDGET (window->grid_direction));
00774       gtk_widget_show (GTK_WIDGET (window->label_step));
00775       gtk_widget_show (GTK_WIDGET (window->spin_step));
00776     }
00777   switch (window_get_direction ())
00778     {
00779     case DIRECTION_METHOD_COORDINATES:
00780       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
00781       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
00782       break;
00783     default:
00784       gtk_widget_show (GTK_WIDGET (window->label_estimates));
00785       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
00786     }
00787 #if DEBUG
00788   fprintf (stderr, "window_update_direction: end\n");
00789 #endif
00790 }
00791
00796 void
00797 window_update ()
00798 {
00799   unsigned int i;
00800 #if DEBUG
00801   fprintf (stderr, "window_update: start\n");
00802 #endif
00803   gtk_widget_set_sensitive
00804     (GTK_WIDGET (window->button_evaluator),
00805      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
00806                                    (window->check_evaluator)));
00807   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
00808   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
00809   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
00810   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
00811   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
00812   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
00813   gtk_widget_hide (GTK_WIDGET (window->label_bests));
00814   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
00815   gtk_widget_hide (GTK_WIDGET (window->label_population));
00816   gtk_widget_hide (GTK_WIDGET (window->spin_population));
00817   gtk_widget_hide (GTK_WIDGET (window->label_generations));
00818   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
00819   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
00820   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
00821   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
00822   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
00823   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
00824   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
00825   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
00826   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
00827   gtk_widget_hide (GTK_WIDGET (window->label_bits));
00828   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
00829   gtk_widget_hide (GTK_WIDGET (window->check_direction));
00830   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
00831   gtk_widget_hide (GTK_WIDGET (window->label_step));
00832   gtk_widget_hide (GTK_WIDGET (window->spin_step));
00833   gtk_widget_hide (GTK_WIDGET (window->label_p));
00834   gtk_widget_hide (GTK_WIDGET (window->spin_p));
00835   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
00836   switch (window_get_algorithm ())
00837     {
00838     case ALGORITHM_MONTE_CARLO:
```

```
00839       gtk_widget_show (GTK_WIDGET (window->label_simulations));
00840       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
00841       gtk_widget_show (GTK_WIDGET (window->label_iterations));
00842       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00843       if (i > 1)
00844         {
00845           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00846           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00847           gtk_widget_show (GTK_WIDGET (window->label_bests));
00848           gtk_widget_show (GTK_WIDGET (window->spin_bests));
00849         }
00850       window_update_direction ();
00851       break;
00852     case ALGORITHM_SWEEP:
00853       gtk_widget_show (GTK_WIDGET (window->label_iterations));
00854       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00855       if (i > 1)
00856         {
00857           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00858           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00859           gtk_widget_show (GTK_WIDGET (window->label_bests));
00860           gtk_widget_show (GTK_WIDGET (window->spin_bests));
00861         }
00862       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
00863       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
00864       gtk_widget_show (GTK_WIDGET (window->check_direction));
00865       window_update_direction ();
00866       break;
00867     default:
00868       gtk_widget_show (GTK_WIDGET (window->label_population));
00869       gtk_widget_show (GTK_WIDGET (window->spin_population));
00870       gtk_widget_show (GTK_WIDGET (window->label_generations));
00871       gtk_widget_show (GTK_WIDGET (window->spin_generations));
00872       gtk_widget_show (GTK_WIDGET (window->label_mutation));
00873       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
00874       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
00875       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
00876       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
00877       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
00878       gtk_widget_show (GTK_WIDGET (window->label_bits));
00879       gtk_widget_show (GTK_WIDGET (window->spin_bits));
00880     }
00881   gtk_widget_set_sensitive
00882     (GTK_WIDGET (window->button_remove_experiment),
      input->nexperiments > 1);
00883   gtk_widget_set_sensitive
00884     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
00885   for (i = 0; i < input->ninputs; ++i)
00886     {
00887       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00888       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00889       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
00890       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
00891       g_signal_handler_block
00892         (window->check_template[i], window->id_template[i]);
00893       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
00894       gtk_toggle_button_set_active
00895         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
00896       g_signal_handler_unblock
00897         (window->button_template[i], window->id_input[i]);
00898       g_signal_handler_unblock
00899         (window->check_template[i], window->id_template[i]);
00900     }
00901   if (i > 0)
00902     {
00903       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
00904       gtk_widget_set_sensitive
00905         (GTK_WIDGET (window->button_template[i - 1]),
00906          gtk_toggle_button_get_active
00907          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
00908     }
00909   if (i < MAX_NINPUTS)
00910     {
00911       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00912       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00913       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
00914       gtk_widget_set_sensitive
00915         (GTK_WIDGET (window->button_template[i]),
00916          gtk_toggle_button_get_active
00917          GTK_TOGGLE_BUTTON (window->check_template[i]));
00918       g_signal_handler_block
00919         (window->check_template[i], window->id_template[i]);
00920       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
00921       gtk_toggle_button_set_active
```

```
00922         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
00923       g_signal_handler_unblock
00924         (window->button_template[i], window->id_input[i]);
00925       g_signal_handler_unblock
00926         (window->check_template[i], window->id_template[i]);
00927     }
00928   while (++i < MAX_NINPUTS)
00929     {
00930       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
00931       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
00932     }
00933   gtk_widget_set_sensitive
00934     (GTK_WIDGET (window->spin_minabs),
00935      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
00936   gtk_widget_set_sensitive
00937     (GTK_WIDGET (window->spin_maxabs),
00938      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
00939   if (window_get_norm () == ERROR_NORM_P)
00940     {
00941       gtk_widget_show (GTK_WIDGET (window->label_p));
00942       gtk_widget_show (GTK_WIDGET (window->spin_p));
00943     }
00944 #if DEBUG
00945   fprintf (stderr, "window_update: end\n");
00946 #endif
00947 }
00948
00953 void
00954 window_set_algorithm ()
00955 {
00956   int i;
00957 #if DEBUG
00958   fprintf (stderr, "window_set_algorithm: start\n");
00959 #endif
00960   i = window_get_algorithm ();
00961   switch (i)
00962     {
00963     case ALGORITHM_SWEEP:
00964       input->nsweeps = (unsigned int *) g_realloc
00965         (input->nsweeps, input->nvariables * sizeof (unsigned int));
00966       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00967       if (i < 0)
00968         i = 0;
00969       gtk_spin_button_set_value (window->spin_sweeps,
00970                                  (gdouble) input->nsweeps[i]);
00971       break;
00972     case ALGORITHM_GENETIC:
00973       input->nbits = (unsigned int *) g_realloc
00974         (input->nbits, input->nvariables * sizeof (unsigned int));
00975       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00976       if (i < 0)
00977         i = 0;
00978       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
00979     }
00980   window_update ();
00981 #if DEBUG
00982   fprintf (stderr, "window_set_algorithm: end\n");
00983 #endif
00984 }
00985
00990 void
00991 window_set_experiment ()
00992 {
00993   unsigned int i, j;
00994   char *buffer1, *buffer2;
00995 #if DEBUG
00996   fprintf (stderr, "window_set_experiment: start\n");
00997 #endif
00998   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
00999   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
01000   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01001   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01002   g_free (buffer1);
01003   g_signal_handler_block
01004     (window->button_experiment, window->id_experiment_name);
01005   gtk_file_chooser_set_filename
01006     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01007   g_signal_handler_unblock
01008     (window->button_experiment, window->id_experiment_name);
01009   g_free (buffer2);
01010   for (j = 0; j < input->ninputs; ++j)
01011     {
01012       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01013       buffer2
01014         = g_build_filename (input->directory, input->template[j][i], NULL);
```

```
01015       gtk_file_chooser_set_filename
01016         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01017       g_free (buffer2);
01018       g_signal_handler_unblock
01019         (window->button_template[j], window->id_input[j]);
01020     }
01021 #if DEBUG
01022   fprintf (stderr, "window_set_experiment: end\n");
01023 #endif
01024 }
01025
01030 void
01031 window_remove_experiment ()
01032 {
01033   unsigned int i, j;
01034 #if DEBUG
01035   fprintf (stderr, "window_remove_experiment: start\n");
01036 #endif
01037   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01038   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01039   gtk_combo_box_text_remove (window->combo_experiment, i);
01040   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01041   xmlFree (input->experiment[i]);
01042   --input->nexperiments;
01043   for (j = i; j < input->nexperiments; ++j)
01044     {
01045       input->experiment[j] = input->experiment[j + 1];
01046       input->weight[j] = input->weight[j + 1];
01047     }
01048   j = input->nexperiments - 1;
01049   if (i > j)
01050     i = j;
01051   for (j = 0; j < input->ninputs; ++j)
01052     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01053   g_signal_handler_block
01054     (window->button_experiment, window->id_experiment_name);
01055   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01056   g_signal_handler_unblock
01057     (window->button_experiment, window->id_experiment_name);
01058   for (j = 0; j < input->ninputs; ++j)
01059     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01060   window_update ();
01061 #if DEBUG
01062   fprintf (stderr, "window_remove_experiment: end\n");
01063 #endif
01064 }
01065
01070 void
01071 window_add_experiment ()
01072 {
01073   unsigned int i, j;
01074 #if DEBUG
01075   fprintf (stderr, "window_add_experiment: start\n");
01076 #endif
01077   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01078   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01079   gtk_combo_box_text_insert_text
01080     (window->combo_experiment, i, input->experiment[i]);
01081   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01082   input->experiment = (char **) g_realloc
01083     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
01084   input->weight = (double *) g_realloc
01085     (input->weight, (input->nexperiments + 1) * sizeof (double));
01086   for (j = input->nexperiments - 1; j > i; --j)
01087     {
01088       input->experiment[j + 1] = input->experiment[j];
01089       input->weight[j + 1] = input->weight[j];
01090     }
01091   input->experiment[j + 1]
01092     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
01093   input->weight[j + 1] = input->weight[j];
01094   ++input->nexperiments;
01095   for (j = 0; j < input->ninputs; ++j)
01096     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01097   g_signal_handler_block
01098     (window->button_experiment, window->id_experiment_name);
01099   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01100   g_signal_handler_unblock
01101     (window->button_experiment, window->id_experiment_name);
01102   for (j = 0; j < input->ninputs; ++j)
```

```
01103     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01104   window_update ();
01105 #if DEBUG
01106   fprintf (stderr, "window_add_experiment: end\n");
01107 #endif
01108 }
01109
01114 void
01115 window_name_experiment ()
01116 {
01117   unsigned int i;
01118   char *buffer;
01119   GFile *file1, *file2;
01120 #if DEBUG
01121   fprintf (stderr, "window_name_experiment: start\n");
01122 #endif
01123   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01124   file1
01125     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01126   file2 = g_file_new_for_path (input->directory);
01127   buffer = g_file_get_relative_path (file2, file1);
01128   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01129   gtk_combo_box_text_remove (window->combo_experiment, i);
01130   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01131   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01132   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01133   g_free (buffer);
01134   g_object_unref (file2);
01135   g_object_unref (file1);
01136 #if DEBUG
01137   fprintf (stderr, "window_name_experiment: end\n");
01138 #endif
01139 }
01140
01145 void
01146 window_weight_experiment ()
01147 {
01148   unsigned int i;
01149 #if DEBUG
01150   fprintf (stderr, "window_weight_experiment: start\n");
01151 #endif
01152   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01153   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
01154 #if DEBUG
01155   fprintf (stderr, "window_weight_experiment: end\n");
01156 #endif
01157 }
01158
01164 void
01165 window_inputs_experiment ()
01166 {
01167   unsigned int j;
01168 #if DEBUG
01169   fprintf (stderr, "window_inputs_experiment: start\n");
01170 #endif
01171   j = input->ninputs - 1;
01172   if (j
01173       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01174                                         (window->check_template[j])))
01175     --input->ninputs;
01176   if (input->ninputs < MAX_NINPUTS
01177       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01178                                        (window->check_template[j])))
01179     {
01180       ++input->ninputs;
01181       for (j = 0; j < input->ninputs; ++j)
01182         {
01183           input->template[j] = (char **)
01184             g_realloc (input->template[j], input->nvariables * sizeof (char *))
      ;
01185         }
01186     }
01187   window_update ();
01188 #if DEBUG
01189   fprintf (stderr, "window_inputs_experiment: end\n");
01190 #endif
01191 }
01192
01200 void
01201 window_template_experiment (void *data)
01202 {
01203   unsigned int i, j;
01204   char *buffer;
01205   GFile *file1, *file2;
```

```
01206 #if DEBUG
01207   fprintf (stderr, "window_template_experiment: start\n");
01208 #endif
01209   i = (size_t) data;
01210   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01211   file1
01212     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01213   file2 = g_file_new_for_path (input->directory);
01214   buffer = g_file_get_relative_path (file2, file1);
01215   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
01216   g_free (buffer);
01217   g_object_unref (file2);
01218   g_object_unref (file1);
01219 #if DEBUG
01220   fprintf (stderr, "window_template_experiment: end\n");
01221 #endif
01222 }
01223
01228 void
01229 window_set_variable ()
01230 {
01231   unsigned int i;
01232 #if DEBUG
01233   fprintf (stderr, "window_set_variable: start\n");
01234 #endif
01235   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01236   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
01237   gtk_entry_set_text (window->entry_variable, input->label[i]);
01238   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
01239   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
01240   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
01241   if (input->rangeminabs[i] != -G_MAXDOUBLE)
01242     {
01243       gtk_spin_button_set_value (window->spin_minabs, input->
    rangeminabs[i]);
01244       gtk_toggle_button_set_active
01245         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01246     }
01247   else
01248     {
01249       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01250       gtk_toggle_button_set_active
01251         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01252     }
01253   if (input->rangemaxabs[i] != G_MAXDOUBLE)
01254     {
01255       gtk_spin_button_set_value (window->spin_maxabs, input->
    rangemaxabs[i]);
01256       gtk_toggle_button_set_active
01257         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01258     }
01259   else
01260     {
01261       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01262       gtk_toggle_button_set_active
01263         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01264     }
01265   gtk_spin_button_set_value (window->spin_precision, input->
    precision[i]);
01266   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
    nsteps);
01267   if (input->nsteps)
01268     gtk_spin_button_set_value (window->spin_step, input->step[i]);
01269 #if DEBUG
01270   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01271            input->precision[i]);
01272 #endif
01273   switch (window_get_algorithm ())
01274     {
01275     case ALGORITHM_SWEEP:
01276       gtk_spin_button_set_value (window->spin_sweeps,
01277                                  (gdouble) input->nsweeps[i]);
01278 #if DEBUG
01279       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01280                input->nsweeps[i]);
01281 #endif
01282       break;
01283     case ALGORITHM_GENETIC:
01284       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
    nbits[i]);
01285 #if DEBUG
01286       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01287                input->nbits[i]);
01288 #endif
01289       break;
```

```
01290     }
01291   window_update ();
01292 #if DEBUG
01293   fprintf (stderr, "window_set_variable: end\n");
01294 #endif
01295 }
01296
01301 void
01302 window_remove_variable ()
01303 {
01304   unsigned int i, j;
01305 #if DEBUG
01306   fprintf (stderr, "window_remove_variable: start\n");
01307 #endif
01308   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01309   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01310   gtk_combo_box_text_remove (window->combo_variable, i);
01311   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01312   xmlFree (input->label[i]);
01313   --input->nvariables;
01314   for (j = i; j < input->nvariables; ++j)
01315     {
01316       input->label[j] = input->label[j + 1];
01317       input->rangemin[j] = input->rangemin[j + 1];
01318       input->rangemax[j] = input->rangemax[j + 1];
01319       input->rangeminabs[j] = input->rangeminabs[j + 1];
01320       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
01321       input->precision[j] = input->precision[j + 1];
01322       input->step[j] = input->step[j + 1];
01323       switch (window_get_algorithm ())
01324         {
01325         case ALGORITHM_SWEEP:
01326           input->nsweeps[j] = input->nsweeps[j + 1];
01327           break;
01328         case ALGORITHM_GENETIC:
01329           input->nbits[j] = input->nbits[j + 1];
01330         }
01331     }
01332   j = input->nvariables - 1;
01333   if (i > j)
01334     i = j;
01335   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01336   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01337   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01338   window_update ();
01339 #if DEBUG
01340   fprintf (stderr, "window_remove_variable: end\n");
01341 #endif
01342 }
01343
01348 void
01349 window_add_variable ()
01350 {
01351   unsigned int i, j;
01352 #if DEBUG
01353   fprintf (stderr, "window_add_variable: start\n");
01354 #endif
01355   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01356   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01357   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
01358   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01359   input->label = (char **) g_realloc
01360     (input->label, (input->nvariables + 1) * sizeof (char *));
01361   input->rangemin = (double *) g_realloc
01362     (input->rangemin, (input->nvariables + 1) * sizeof (double));
01363   input->rangemax = (double *) g_realloc
01364     (input->rangemax, (input->nvariables + 1) * sizeof (double));
01365   input->rangeminabs = (double *) g_realloc
01366     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
01367   input->rangemaxabs = (double *) g_realloc
01368     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
01369   input->precision = (unsigned int *) g_realloc
01370     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
01371   input->step = (double *) g_realloc
01372     (input->step, (input->nvariables + 1) * sizeof (double));
01373   for (j = input->nvariables - 1; j > i; --j)
01374     {
01375       input->label[j + 1] = input->label[j];
01376       input->rangemin[j + 1] = input->rangemin[j];
01377       input->rangemax[j + 1] = input->rangemax[j];
```

```
01378       input->rangeminabs[j + 1] = input->rangeminabs[j];
01379       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
01380       input->precision[j + 1] = input->precision[j];
01381       input->step[j + 1] = input->step[j];
01382     }
01383   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->
      label[j]);
01384   input->rangemin[j + 1] = input->rangemin[j];
01385   input->rangemax[j + 1] = input->rangemax[j];
01386   input->rangeminabs[j + 1] = input->rangeminabs[j];
01387   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
01388   input->precision[j + 1] = input->precision[j];
01389   input->step[j + 1] = input->step[j];
01390   switch (window_get_algorithm ())
01391     {
01392     case ALGORITHM_SWEEP:
01393       input->nsweeps = (unsigned int *) g_realloc
01394         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
01395       for (j = input->nvariables - 1; j > i; --j)
01396         input->nsweeps[j + 1] = input->nsweeps[j];
01397       input->nsweeps[j + 1] = input->nsweeps[j];
01398       break;
01399     case ALGORITHM_GENETIC:
01400       input->nbits = (unsigned int *) g_realloc
01401         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
01402       for (j = input->nvariables - 1; j > i; --j)
01403         input->nbits[j + 1] = input->nbits[j];
01404       input->nbits[j + 1] = input->nbits[j];
01405     }
01406   ++input->nvariables;
01407   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01408   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01409   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01410   window_update ();
01411 #if DEBUG
01412   fprintf (stderr, "window_add_variable: end\n");
01413 #endif
01414 }
01415
01420 void
01421 window_label_variable ()
01422 {
01423   unsigned int i;
01424   const char *buffer;
01425 #if DEBUG
01426   fprintf (stderr, "window_label_variable: start\n");
01427 #endif
01428   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01429   buffer = gtk_entry_get_text (window->entry_variable);
01430   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01431   gtk_combo_box_text_remove (window->combo_variable, i);
01432   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01433   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01434   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01435 #if DEBUG
01436   fprintf (stderr, "window_label_variable: end\n");
01437 #endif
01438 }
01439
01444 void
01445 window_precision_variable ()
01446 {
01447   unsigned int i;
01448 #if DEBUG
01449   fprintf (stderr, "window_precision_variable: start\n");
01450 #endif
01451   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01452   input->precision[i]
01453     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01454   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
01455   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
01456   gtk_spin_button_set_digits (window->spin_minabs, input->
      precision[i]);
01457   gtk_spin_button_set_digits (window->spin_maxabs, input->
      precision[i]);
01458 #if DEBUG
01459   fprintf (stderr, "window_precision_variable: end\n");
01460 #endif
01461 }
01462
01467 void
01468 window_rangemin_variable ()
01469 {
```

```
01470   unsigned int i;
01471 #if DEBUG
01472   fprintf (stderr, "window_rangemin_variable: start\n");
01473 #endif
01474   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01475   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
01476 #if DEBUG
01477   fprintf (stderr, "window_rangemin_variable: end\n");
01478 #endif
01479 }
01480
01485 void
01486 window_rangemax_variable ()
01487 {
01488   unsigned int i;
01489 #if DEBUG
01490   fprintf (stderr, "window_rangemax_variable: start\n");
01491 #endif
01492   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01493   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
01494 #if DEBUG
01495   fprintf (stderr, "window_rangemax_variable: end\n");
01496 #endif
01497 }
01498
01503 void
01504 window_rangeminabs_variable ()
01505 {
01506   unsigned int i;
01507 #if DEBUG
01508   fprintf (stderr, "window_rangeminabs_variable: start\n");
01509 #endif
01510   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01511   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
01512 #if DEBUG
01513   fprintf (stderr, "window_rangeminabs_variable: end\n");
01514 #endif
01515 }
01516
01521 void
01522 window_rangemaxabs_variable ()
01523 {
01524   unsigned int i;
01525 #if DEBUG
01526   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01527 #endif
01528   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01529   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
01530 #if DEBUG
01531   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01532 #endif
01533 }
01534
01539 void
01540 window_step_variable ()
01541 {
01542   unsigned int i;
01543 #if DEBUG
01544   fprintf (stderr, "window_step_variable: start\n");
01545 #endif
01546   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01547   input->step[i] = gtk_spin_button_get_value (window->spin_step);
01548 #if DEBUG
01549   fprintf (stderr, "window_step_variable: end\n");
01550 #endif
01551 }
01552
01557 void
01558 window_update_variable ()
01559 {
01560   int i;
01561 #if DEBUG
01562   fprintf (stderr, "window_update_variable: start\n");
01563 #endif
01564   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01565   if (i < 0)
01566     i = 0;
01567   switch (window_get_algorithm ())
01568     {
01569     case ALGORITHM_SWEEP:
01570       input->nsweeps[i]
01571         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01572 #if DEBUG
01573       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01574                input->nsweeps[i]);
```

```
01575 #endif
01576       break;
01577     case ALGORITHM_GENETIC:
01578       input->nbits[i] = gtk_spin_button_get_value_as_int (window->
      spin_bits);
01579 #if DEBUG
01580       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01581                input->nbits[i]);
01582 #endif
01583     }
01584 #if DEBUG
01585   fprintf (stderr, "window_update_variable: end\n");
01586 #endif
01587 }
01588
01596 int
01597 window_read (char *filename)
01598 {
01599   unsigned int i;
01600   char *buffer;
01601 #if DEBUG
01602   fprintf (stderr, "window_read: start\n");
01603 #endif
01604
01605   // Reading new input file
01606   input_free ();
01607   if (!input_open (filename))
01608     return 0;
01609
01610   // Setting GTK+ widgets data
01611   gtk_entry_set_text (window->entry_result, input->result);
01612   gtk_entry_set_text (window->entry_variables, input->
      variables);
01613   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01614   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01615                                  (window->button_simulator), buffer);
01616   g_free (buffer);
01617   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01618                                 (size_t) input->evaluator);
01619   if (input->evaluator)
01620     {
01621       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01622       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01623                                      (window->button_evaluator), buffer);
01624       g_free (buffer);
01625     }
01626   gtk_toggle_button_set_active
01627     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01628   switch (input->algorithm)
01629     {
01630     case ALGORITHM_MONTE_CARLO:
01631       gtk_spin_button_set_value (window->spin_simulations,
01632                                  (gdouble) input->nsimulations);
01633     case ALGORITHM_SWEEP:
01634       gtk_spin_button_set_value (window->spin_iterations,
01635                                  (gdouble) input->niterations);
01636       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01637       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01638       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
01639                                     input->nsteps);
01640       if (input->nsteps)
01641         {
01642           gtk_toggle_button_set_active
01643             (GTK_TOGGLE_BUTTON (window->button_direction
01644                                 [input->direction]), TRUE);
01645           gtk_spin_button_set_value (window->spin_steps,
01646                                      (gdouble) input->nsteps);
01647           gtk_spin_button_set_value (window->spin_relaxation,
01648                                      (gdouble) input->relaxation);
01649           switch (input->direction)
01650             {
01651             case DIRECTION_METHOD_RANDOM:
01652               gtk_spin_button_set_value (window->spin_estimates,
01653                                          (gdouble) input->nestimates);
01654             }
01655         }
01656       break;
01657     default:
01658       gtk_spin_button_set_value (window->spin_population,
01659                                  (gdouble) input->nsimulations);
01660       gtk_spin_button_set_value (window->spin_generations,
01661                                  (gdouble) input->niterations);
```

```
01662       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01663       gtk_spin_button_set_value (window->spin_reproduction,
01664                                  input->reproduction_ratio);
01665       gtk_spin_button_set_value (window->spin_adaptation,
01666                                  input->adaptation_ratio);
01667     }
01668   gtk_toggle_button_set_active
01669     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01670   gtk_spin_button_set_value (window->spin_p, input->p);
01671   gtk_spin_button_set_value (window->spin_thresold, input->
      thresold);
01672   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01673   g_signal_handler_block (window->button_experiment,
01674                           window->id_experiment_name);
01675   gtk_combo_box_text_remove_all (window->combo_experiment);
01676   for (i = 0; i < input->nexperiments; ++i)
01677     gtk_combo_box_text_append_text (window->combo_experiment,
01678                                     input->experiment[i]);
01679   g_signal_handler_unblock
01680     (window->button_experiment, window->id_experiment_name);
01681   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01682   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01683   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01684   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01685   gtk_combo_box_text_remove_all (window->combo_variable);
01686   for (i = 0; i < input->nvariables; ++i)
01687     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
01688   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01689   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01690   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01691   window_set_variable ();
01692   window_update ();
01693
01694 #if DEBUG
01695   fprintf (stderr, "window_read: end\n");
01696 #endif
01697   return 1;
01698 }
01699
01704 void
01705 window_open ()
01706 {
01707   GtkFileChooserDialog *dlg;
01708   GtkFileFilter *filter;
01709   char *buffer, *directory, *name;
01710
01711 #if DEBUG
01712   fprintf (stderr, "window_open: start\n");
01713 #endif
01714
01715   // Saving a backup of the current input file
01716   directory = g_strdup (input->directory);
01717   name = g_strdup (input->name);
01718
01719   // Opening dialog
01720   dlg = (GtkFileChooserDialog *)
01721     gtk_file_chooser_dialog_new (gettext ("Open input file"),
01722                                  window->window,
01723                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01724                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
01725                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
01726
01727   // Adding XML filter
01728   filter = (GtkFileFilter *) gtk_file_filter_new ();
01729   gtk_file_filter_set_name (filter, "XML");
01730   gtk_file_filter_add_pattern (filter, "*.xml");
01731   gtk_file_filter_add_pattern (filter, "*.XML");
01732   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
01733
01734   // If OK saving
01735   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
01736     {
01737
01738       // Traying to open the input file
01739       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
01740       if (!window_read (buffer))
01741         {
01742 #if DEBUG
01743           fprintf (stderr, "window_open: error reading input file\n");
```

```
01744 #endif
01745           g_free (buffer);
01746
01747           // Reading backup file on error
01748           buffer = g_build_filename (directory, name, NULL);
01749           if (!input_open (buffer))
01750             {
01751
01752               // Closing on backup file reading error
01753 #if DEBUG
01754               fprintf (stderr, "window_read: error reading backup file\n");
01755 #endif
01756               g_free (buffer);
01757               break;
01758             }
01759           g_free (buffer);
01760         }
01761       else
01762         {
01763           g_free (buffer);
01764           break;
01765         }
01766     }
01767
01768   // Freeing and closing
01769   g_free (name);
01770   g_free (directory);
01771   gtk_widget_destroy (GTK_WIDGET (dlg));
01772 #if DEBUG
01773   fprintf (stderr, "window_open: end\n");
01774 #endif
01775 }
01776
01781 void
01782 window_new ()
01783 {
01784   unsigned int i;
01785   char *buffer, *buffer2, buffer3[64];
01786   char *label_algorithm[NALGORITHMS] = {
01787     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
01788   };
01789   char *tip_algorithm[NALGORITHMS] = {
01790     gettext ("Monte-Carlo brute force algorithm"),
01791     gettext ("Sweep brute force algorithm"),
01792     gettext ("Genetic algorithm")
01793   };
01794   char *label_direction[NDIRECTIONS] = {
01795     gettext ("_Coordinates descent"), gettext ("_Random")
01796   };
01797   char *tip_direction[NDIRECTIONS] = {
01798     gettext ("Coordinates direction estimate method"),
01799     gettext ("Random direction estimate method")
01800   };
01801   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
01802   char *tip_norm[NNORMS] = {
01803     gettext ("Euclidean error norm (L2)"),
01804     gettext ("Maximum error norm (L)"),
01805     gettext ("P error norm (Lp)"),
01806     gettext ("Taxicab error norm (L1)")
01807   };
01808
01809 #if DEBUG
01810   fprintf (stderr, "window_new: start\n");
01811 #endif
01812
01813   // Creating the window
01814   window->window = main_window
01815     = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
01816
01817   // Finish when closing the window
01818   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
01819
01820   // Setting the window title
01821   gtk_window_set_title (window->window, "MPCOTool");
01822
01823   // Creating the open button
01824   window->button_open = (GtkToolButton *) gtk_tool_button_new
01825     (gtk_image_new_from_icon_name ("document-open",
01826                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01827      gettext ("Open"));
01828   g_signal_connect (window->button_open, "clicked", window_open, NULL);
01829
01830   // Creating the save button
01831   window->button_save = (GtkToolButton *) gtk_tool_button_new
01832     (gtk_image_new_from_icon_name ("document-save",
01833                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01834      gettext ("Save"));
```

```
01835   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
01836                     NULL);
01837
01838   // Creating the run button
01839   window->button_run = (GtkToolButton *) gtk_tool_button_new
01840     (gtk_image_new_from_icon_name ("system-run",
01841                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01842      gettext ("Run"));
01843   g_signal_connect (window->button_run, "clicked", window_run, NULL);
01844
01845   // Creating the options button
01846   window->button_options = (GtkToolButton *) gtk_tool_button_new
01847     (gtk_image_new_from_icon_name ("preferences-system",
01848                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01849      gettext ("Options"));
01850   g_signal_connect (window->button_options, "clicked", options_new, NULL);
01851
01852   // Creating the help button
01853   window->button_help = (GtkToolButton *) gtk_tool_button_new
01854     (gtk_image_new_from_icon_name ("help-browser",
01855                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01856      gettext ("Help"));
01857   g_signal_connect (window->button_help, "clicked", window_help, NULL);
01858
01859   // Creating the about button
01860   window->button_about = (GtkToolButton *) gtk_tool_button_new
01861     (gtk_image_new_from_icon_name ("help-about",
01862                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01863      gettext ("About"));
01864   g_signal_connect (window->button_about, "clicked", window_about, NULL);
01865
01866   // Creating the exit button
01867   window->button_exit = (GtkToolButton *) gtk_tool_button_new
01868     (gtk_image_new_from_icon_name ("application-exit",
01869                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01870      gettext ("Exit"));
01871   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
01872
01873   // Creating the buttons bar
01874   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
01875   gtk_toolbar_insert
01876     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
01877   gtk_toolbar_insert
01878     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
01879   gtk_toolbar_insert
01880     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
01881   gtk_toolbar_insert
01882     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
01883   gtk_toolbar_insert
01884     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
01885   gtk_toolbar_insert
01886     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
01887   gtk_toolbar_insert
01888     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
01889   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
01890
01891   // Creating the simulator program label and entry
01892   window->label_simulator
01893     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
01894   window->button_simulator = (GtkFileChooserButton *)
01895     gtk_file_chooser_button_new (gettext ("Simulator program"),
01896                                  GTK_FILE_CHOOSER_ACTION_OPEN);
01897   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
01898                                gettext ("Simulator program executable file"));
01899   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
01900
01901   // Creating the evaluator program label and entry
01902   window->check_evaluator = (GtkCheckButton *)
01903     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
01904   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
01905   window->button_evaluator = (GtkFileChooserButton *)
01906     gtk_file_chooser_button_new (gettext ("Evaluator program"),
01907                                  GTK_FILE_CHOOSER_ACTION_OPEN);
01908   gtk_widget_set_tooltip_text
01909     (GTK_WIDGET (window->button_evaluator),
01910      gettext ("Optional evaluator program executable file"));
01911
01912   // Creating the results files labels and entries
01913   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
01914   window->entry_result = (GtkEntry *) gtk_entry_new ();
01915   gtk_widget_set_tooltip_text
01916     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
01917   window->label_variables
01918     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
01919   window->entry_variables = (GtkEntry *) gtk_entry_new ();
```

```
01920   gtk_widget_set_tooltip_text
01921     (GTK_WIDGET (window->entry_variables),
01922      gettext ("All simulated results file"));
01923
01924   // Creating the files grid and attaching widgets
01925   window->grid_files = (GtkGrid *) gtk_grid_new ();
01926   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
01927                    0, 0, 1, 1);
01928   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
01929                    1, 0, 1, 1);
01930   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
01931                    0, 1, 1, 1);
01932   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
01933                    1, 1, 1, 1);
01934   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
01935                    0, 2, 1, 1);
01936   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
01937                    1, 2, 1, 1);
01938   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
01939                    0, 3, 1, 1);
01940   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
01941                    1, 3, 1, 1);
01942
01943   // Creating the algorithm properties
01944   window->label_simulations = (GtkLabel *) gtk_label_new
01945     (gettext ("Simulations number"));
01946   window->spin_simulations
01947     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01948   gtk_widget_set_tooltip_text
01949     (GTK_WIDGET (window->spin_simulations),
01950      gettext ("Number of simulations to perform for each iteration"));
01951   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
01952   window->label_iterations = (GtkLabel *)
01953     gtk_label_new (gettext ("Iterations number"));
01954   window->spin_iterations
01955     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01956   gtk_widget_set_tooltip_text
01957     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
01958   g_signal_connect
01959     (window->spin_iterations, "value-changed", window_update, NULL);
01960   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
01961   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
01962   window->spin_tolerance
01963     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01964   gtk_widget_set_tooltip_text
01965     (GTK_WIDGET (window->spin_tolerance),
01966      gettext ("Tolerance to set the variable interval on the next iteration"));
01967   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
01968   window->spin_bests
01969     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01970   gtk_widget_set_tooltip_text
01971     (GTK_WIDGET (window->spin_bests),
01972      gettext ("Number of best simulations used to set the variable interval "
01973               "on the next iteration"));
01974   window->label_population
01975     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
01976   window->spin_population
01977     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01978   gtk_widget_set_tooltip_text
01979     (GTK_WIDGET (window->spin_population),
01980      gettext ("Number of population for the genetic algorithm"));
01981   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
01982   window->label_generations
01983     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
01984   window->spin_generations
01985     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01986   gtk_widget_set_tooltip_text
01987     (GTK_WIDGET (window->spin_generations),
01988      gettext ("Number of generations for the genetic algorithm"));
01989   window->label_mutation
01990     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
01991   window->spin_mutation
01992     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01993   gtk_widget_set_tooltip_text
01994     (GTK_WIDGET (window->spin_mutation),
01995      gettext ("Ratio of mutation for the genetic algorithm"));
01996   window->label_reproduction
01997     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
01998   window->spin_reproduction
```

```
01999     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02000   gtk_widget_set_tooltip_text
02001     (GTK_WIDGET (window->spin_reproduction),
02002      gettext ("Ratio of reproduction for the genetic algorithm"));
02003   window->label_adaptation
02004     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
02005   window->spin_adaptation
02006     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02007   gtk_widget_set_tooltip_text
02008     (GTK_WIDGET (window->spin_adaptation),
02009      gettext ("Ratio of adaptation for the genetic algorithm"));
02010   window->label_thresold = (GtkLabel *) gtk_label_new (gettext ("Thresold"));
02011   window->spin_thresold = (GtkSpinButton *) gtk_spin_button_new_with_range
02012     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02013   gtk_widget_set_tooltip_text
02014     (GTK_WIDGET (window->spin_thresold),
02015      gettext ("Thresold in the objective function to finish the simulations"));
02016   window->scrolled_thresold
02017     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02018   gtk_container_add (GTK_CONTAINER (window->scrolled_thresold),
02019                      GTK_WIDGET (window->spin_thresold));
02020 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_thresold), TRUE);
02021 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_thresold),
02022 //                         GTK_ALIGN_FILL);
02023
02024   // Creating the direction search method properties
02025   window->check_direction = (GtkCheckButton *)
02026     gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
02027   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02028   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02029   window->button_direction[0] = (GtkRadioButton *)
02030     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02031   gtk_grid_attach (window->grid_direction,
02032                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02033   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02034                     NULL);
02035   for (i = 0; ++i < NDIRECTIONS;)
02036     {
02037       window->button_direction[i] = (GtkRadioButton *)
02038         gtk_radio_button_new_with_mnemonic
02039         (gtk_radio_button_get_group (window->button_direction[0]),
02040          label_direction[i]);
02041       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02042                                    tip_direction[i]);
02043       gtk_grid_attach (window->grid_direction,
02044                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02045       g_signal_connect (window->button_direction[i], "clicked",
02046                         window_update, NULL);
02047     }
02048   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
02049   window->spin_steps = (GtkSpinButton *)
02050     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02051   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02052   window->label_estimates
02053     = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
02054   window->spin_estimates = (GtkSpinButton *)
02055     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02056   window->label_relaxation
02057     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
02058   window->spin_relaxation = (GtkSpinButton *)
02059     gtk_spin_button_new_with_range (0., 2., 0.001);
02060   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02061                    0, NDIRECTIONS, 1, 1);
02062   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02063                    1, NDIRECTIONS, 1, 1);
02064   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_estimates),
02065                    0, NDIRECTIONS + 1, 1, 1);
02066   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_estimates),
02067                    1, NDIRECTIONS + 1, 1, 1);
02068   gtk_grid_attach (window->grid_direction,
02069                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
02070                    1);
02071   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_relaxation),
02072                    1, NDIRECTIONS + 2, 1, 1);
02073
02074   // Creating the array of algorithms
02075   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02076   window->button_algorithm[0] = (GtkRadioButton *)
02077     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02078   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
```

```
02079                                tip_algorithm[0]);
02080   gtk_grid_attach (window->grid_algorithm,
02081                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02082   g_signal_connect (window->button_algorithm[0], "clicked",
02083                     window_set_algorithm, NULL);
02084   for (i = 0; ++i < NALGORITHMS;)
02085     {
02086       window->button_algorithm[i] = (GtkRadioButton *)
02087         gtk_radio_button_new_with_mnemonic
02088         (gtk_radio_button_get_group (window->button_algorithm[0]),
02089          label_algorithm[i]);
02090       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02091                                    tip_algorithm[i]);
02092       gtk_grid_attach (window->grid_algorithm,
02093                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02094       g_signal_connect (window->button_algorithm[i], "clicked",
02095                         window_set_algorithm, NULL);
02096     }
02097   gtk_grid_attach (window->grid_algorithm,
02098                    GTK_WIDGET (window->label_simulations), 0,
02099                    NALGORITHMS, 1, 1);
02100   gtk_grid_attach (window->grid_algorithm,
02101                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02102   gtk_grid_attach (window->grid_algorithm,
02103                    GTK_WIDGET (window->label_iterations), 0,
02104                    NALGORITHMS + 1, 1, 1);
02105   gtk_grid_attach (window->grid_algorithm,
02106                    GTK_WIDGET (window->spin_iterations), 1,
02107                    NALGORITHMS + 1, 1, 1);
02108   gtk_grid_attach (window->grid_algorithm,
02109                    GTK_WIDGET (window->label_tolerance), 0,
02110                    NALGORITHMS + 2, 1, 1);
02111   gtk_grid_attach (window->grid_algorithm,
02112                    GTK_WIDGET (window->spin_tolerance), 1,
02113                    NALGORITHMS + 2, 1, 1);
02114   gtk_grid_attach (window->grid_algorithm,
02115                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02116   gtk_grid_attach (window->grid_algorithm,
02117                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02118   gtk_grid_attach (window->grid_algorithm,
02119                    GTK_WIDGET (window->label_population), 0,
02120                    NALGORITHMS + 4, 1, 1);
02121   gtk_grid_attach (window->grid_algorithm,
02122                    GTK_WIDGET (window->spin_population), 1,
02123                    NALGORITHMS + 4, 1, 1);
02124   gtk_grid_attach (window->grid_algorithm,
02125                    GTK_WIDGET (window->label_generations), 0,
02126                    NALGORITHMS + 5, 1, 1);
02127   gtk_grid_attach (window->grid_algorithm,
02128                    GTK_WIDGET (window->spin_generations), 1,
02129                    NALGORITHMS + 5, 1, 1);
02130   gtk_grid_attach (window->grid_algorithm,
02131                    GTK_WIDGET (window->label_mutation), 0,
02132                    NALGORITHMS + 6, 1, 1);
02133   gtk_grid_attach (window->grid_algorithm,
02134                    GTK_WIDGET (window->spin_mutation), 1,
02135                    NALGORITHMS + 6, 1, 1);
02136   gtk_grid_attach (window->grid_algorithm,
02137                    GTK_WIDGET (window->label_reproduction), 0,
02138                    NALGORITHMS + 7, 1, 1);
02139   gtk_grid_attach (window->grid_algorithm,
02140                    GTK_WIDGET (window->spin_reproduction), 1,
02141                    NALGORITHMS + 7, 1, 1);
02142   gtk_grid_attach (window->grid_algorithm,
02143                    GTK_WIDGET (window->label_adaptation), 0,
02144                    NALGORITHMS + 8, 1, 1);
02145   gtk_grid_attach (window->grid_algorithm,
02146                    GTK_WIDGET (window->spin_adaptation), 1,
02147                    NALGORITHMS + 8, 1, 1);
02148   gtk_grid_attach (window->grid_algorithm,
02149                    GTK_WIDGET (window->check_direction), 0,
02150                    NALGORITHMS + 9, 2, 1);
02151   gtk_grid_attach (window->grid_algorithm,
02152                    GTK_WIDGET (window->grid_direction), 0,
02153                    NALGORITHMS + 10, 2, 1);
02154   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
     label_thresold),
02155                    0, NALGORITHMS + 11, 1, 1);
02156   gtk_grid_attach (window->grid_algorithm,
02157                    GTK_WIDGET (window->scrolled_thresold), 1,
02158                    NALGORITHMS + 11, 1, 1);
02159   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02160   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02161                      GTK_WIDGET (window->grid_algorithm));
02162
02163   // Creating the variable widgets
02164   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
```

```
02165   gtk_widget_set_tooltip_text
02166     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02167   window->id_variable = g_signal_connect
02168     (window->combo_variable, "changed", window_set_variable, NULL);
02169   window->button_add_variable
02170     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02171                                                    GTK_ICON_SIZE_BUTTON);
02172   g_signal_connect
02173     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02174   gtk_widget_set_tooltip_text
02175     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02176   window->button_remove_variable
02177     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02178                                                    GTK_ICON_SIZE_BUTTON);
02179   g_signal_connect
02180     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02181   gtk_widget_set_tooltip_text
02182     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02183   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02184   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02185   gtk_widget_set_tooltip_text
02186     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02187   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02188   window->id_variable_label = g_signal_connect
02189     (window->entry_variable, "changed", window_label_variable, NULL);
02190   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02191   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02192     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02193   gtk_widget_set_tooltip_text
02194     (GTK_WIDGET (window->spin_min),
02195      gettext ("Minimum initial value of the variable"));
02196   window->scrolled_min
02197     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02198   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02199                      GTK_WIDGET (window->spin_min));
02200   g_signal_connect (window->spin_min, "value-changed",
02201                     window_rangemin_variable, NULL);
02202   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02203   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02204     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02205   gtk_widget_set_tooltip_text
02206     (GTK_WIDGET (window->spin_max),
02207      gettext ("Maximum initial value of the variable"));
02208   window->scrolled_max
02209     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02210   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02211                      GTK_WIDGET (window->spin_max));
02212   g_signal_connect (window->spin_max, "value-changed",
02213                     window_rangemax_variable, NULL);
02214   window->check_minabs = (GtkCheckButton *)
02215     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
02216   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02217   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02218     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02219   gtk_widget_set_tooltip_text
02220     (GTK_WIDGET (window->spin_minabs),
02221      gettext ("Minimum allowed value of the variable"));
02222   window->scrolled_minabs
02223     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02224   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02225                      GTK_WIDGET (window->spin_minabs));
02226   g_signal_connect (window->spin_minabs, "value-changed",
02227                     window_rangeminabs_variable, NULL);
02228   window->check_maxabs = (GtkCheckButton *)
02229     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
02230   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02231   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02232     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02233   gtk_widget_set_tooltip_text
02234     (GTK_WIDGET (window->spin_maxabs),
02235      gettext ("Maximum allowed value of the variable"));
02236   window->scrolled_maxabs
02237     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02238   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02239                      GTK_WIDGET (window->spin_maxabs));
02240   g_signal_connect (window->spin_maxabs, "value-changed",
02241                     window_rangemaxabs_variable, NULL);
02242   window->label_precision
02243     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02244   window->spin_precision = (GtkSpinButton *)
02245     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02246   gtk_widget_set_tooltip_text
02247     (GTK_WIDGET (window->spin_precision),
02248      gettext ("Number of precision floating point digits\n"
02249               "0 is for integer numbers"));
```

```
02250   g_signal_connect (window->spin_precision, "value-changed",
02251                     window_precision_variable, NULL);
02252   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02253   window->spin_sweeps
02254     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02255   gtk_widget_set_tooltip_text
02256     (GTK_WIDGET (window->spin_sweeps),
02257      gettext ("Number of steps sweeping the variable"));
02258   g_signal_connect
02259     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02260   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02261   window->spin_bits
02262     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02263   gtk_widget_set_tooltip_text
02264     (GTK_WIDGET (window->spin_bits),
02265      gettext ("Number of bits to encode the variable"));
02266   g_signal_connect
02267     (window->spin_bits, "value-changed", window_update_variable, NULL);
02268   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02269   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02270     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02271   gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_step),
02273      gettext ("Initial step size for the direction search method"));
02274   window->scrolled_step
02275     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02276   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02277                      GTK_WIDGET (window->spin_step));
02278   g_signal_connect
02279     (window->spin_step, "value-changed", window_step_variable, NULL);
02280   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02281   gtk_grid_attach (window->grid_variable,
02282                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02283   gtk_grid_attach (window->grid_variable,
02284                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02285   gtk_grid_attach (window->grid_variable,
02286                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02287   gtk_grid_attach (window->grid_variable,
02288                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02289   gtk_grid_attach (window->grid_variable,
02290                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02291   gtk_grid_attach (window->grid_variable,
02292                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02293   gtk_grid_attach (window->grid_variable,
02294                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02295   gtk_grid_attach (window->grid_variable,
02296                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02297   gtk_grid_attach (window->grid_variable,
02298                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02299   gtk_grid_attach (window->grid_variable,
02300                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02301   gtk_grid_attach (window->grid_variable,
02302                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02303   gtk_grid_attach (window->grid_variable,
02304                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02305   gtk_grid_attach (window->grid_variable,
02306                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02307   gtk_grid_attach (window->grid_variable,
02308                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02309   gtk_grid_attach (window->grid_variable,
02310                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02311   gtk_grid_attach (window->grid_variable,
02312                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02313   gtk_grid_attach (window->grid_variable,
02314                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02315   gtk_grid_attach (window->grid_variable,
02316                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02317   gtk_grid_attach (window->grid_variable,
02318                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02319   gtk_grid_attach (window->grid_variable,
02320                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02321   gtk_grid_attach (window->grid_variable,
02322                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02323   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02324   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02325                      GTK_WIDGET (window->grid_variable));
02326
02327   // Creating the experiment widgets
02328   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02329   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02330                                gettext ("Experiment selector"));
02331   window->id_experiment = g_signal_connect
02332     (window->combo_experiment, "changed", window_set_experiment, NULL)
    ;
02333   window->button_add_experiment
02334     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02335                                                    GTK_ICON_SIZE_BUTTON);
```

```
02336   g_signal_connect
02337     (window->button_add_experiment, "clicked",
    window_add_experiment, NULL);
02338   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02339                                gettext ("Add experiment"));
02340   window->button_remove_experiment
02341     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02342                                                    GTK_ICON_SIZE_BUTTON);
02343   g_signal_connect (window->button_remove_experiment, "clicked",
02344                     window_remove_experiment, NULL);
02345   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02346                                gettext ("Remove experiment"));
02347   window->label_experiment
02348     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02349   window->button_experiment = (GtkFileChooserButton *)
02350     gtk_file_chooser_button_new (gettext ("Experimental data file"),
02351                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02352   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02353                                gettext ("Experimental data file"));
02354   window->id_experiment_name
02355     = g_signal_connect (window->button_experiment, "selection-changed",
02356                         window_name_experiment, NULL);
02357   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02358   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02359   window->spin_weight
02360     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02361   gtk_widget_set_tooltip_text
02362     (GTK_WIDGET (window->spin_weight),
02363      gettext ("Weight factor to build the objective function"));
02364   g_signal_connect
02365     (window->spin_weight, "value-changed", window_weight_experiment,
    NULL);
02366   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02367   gtk_grid_attach (window->grid_experiment,
02368                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02369   gtk_grid_attach (window->grid_experiment,
02370                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02371   gtk_grid_attach (window->grid_experiment,
02372                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02373   gtk_grid_attach (window->grid_experiment,
02374                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02375   gtk_grid_attach (window->grid_experiment,
02376                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02377   gtk_grid_attach (window->grid_experiment,
02378                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02379   gtk_grid_attach (window->grid_experiment,
02380                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02381   for (i = 0; i < MAX_NINPUTS; ++i)
02382     {
02383       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02384       window->check_template[i] = (GtkCheckButton *)
02385         gtk_check_button_new_with_label (buffer3);
02386       window->id_template[i]
02387         = g_signal_connect (window->check_template[i], "toggled",
02388                             window_inputs_experiment, NULL);
02389       gtk_grid_attach (window->grid_experiment,
02390                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02391       window->button_template[i] = (GtkFileChooserButton *)
02392         gtk_file_chooser_button_new (gettext ("Input template"),
02393                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02394       gtk_widget_set_tooltip_text
02395         (GTK_WIDGET (window->button_template[i]),
02396          gettext ("Experimental input template file"));
02397       window->id_input[i]
02398         = g_signal_connect_swapped (window->button_template[i],
02399                                     "selection-changed",
02400                                     (void (*)) window_template_experiment,
02401                                     (void *) (size_t) i);
02402       gtk_grid_attach (window->grid_experiment,
02403                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02404     }
02405   window->frame_experiment
02406     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02407   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02408                      GTK_WIDGET (window->grid_experiment));
02409
02410   // Creating the error norm widgets
02411   window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02412   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02413   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02414                      GTK_WIDGET (window->grid_norm));
02415   window->button_norm[0] = (GtkRadioButton *)
02416     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02417   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02418                                tip_norm[0]);
02419   gtk_grid_attach (window->grid_norm,
02420                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
```

```
02421   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02422   for (i = 0; ++i < NNORMS;)
02423     {
02424       window->button_norm[i] = (GtkRadioButton *)
02425         gtk_radio_button_new_with_mnemonic
02426         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02427       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02428                                    tip_norm[i]);
02429       gtk_grid_attach (window->grid_norm,
02430                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02431       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02432     }
02433   window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02434   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02435   window->spin_p = (GtkSpinButton *)
02436     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02437   gtk_widget_set_tooltip_text
02438     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02439   window->scrolled_p
02440     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02441   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02442                      GTK_WIDGET (window->spin_p));
02443   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02444   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02445   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02446                    1, 2, 1, 2);
02447
02448   // Creating the grid and attaching the widgets to the grid
02449   window->grid = (GtkGrid *) gtk_grid_new ();
02450   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02451   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02452   gtk_grid_attach (window->grid,
02453                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02454   gtk_grid_attach (window->grid,
02455                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02456   gtk_grid_attach (window->grid,
02457                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02458   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02459   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
02460
02461   // Setting the window logo
02462   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02463   gtk_window_set_icon (window->window, window->logo);
02464
02465   // Showing the window
02466   gtk_widget_show_all (GTK_WIDGET (window->window));
02467
02468   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02469 #if GTK_MINOR_VERSION >= 16
02470   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02471   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02472   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02473   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02474   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02475   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02476   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_thresold), -1, 40);
02477 #endif
02478
02479   // Reading initial example
02480   input_new ();
02481   buffer2 = g_get_current_dir ();
02482   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02483   g_free (buffer2);
02484   window_read (buffer);
02485   g_free (buffer);
02486
02487 #if DEBUG
02488   fprintf (stderr, "window_new: start\n");
02489 #endif
02490 }
```

5.5 interface.h File Reference

Header file to define the graphical interface functions.

This graph shows which files directly or indirectly include this file:

Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.
- void input_save (char *filename)

  Function to save the input file.
- void options_new ()

  Function to open the options dialog.
- void running_new ()

  Function to open the running dialog.
- unsigned int window_get_algorithm ()

  Function to get the stochastic algorithm number.
- unsigned int window_get_direction ()

  Function to get the direction search method number.
- unsigned int window_get_norm ()

  Function to get the norm method number.
- void window_save_direction ()

  Function to save the direction search method data in the input file.
- int window_save ()

  Function to save the input file.
- void window_run ()

  Function to run a optimization.
- void window_help ()

  Function to show a help dialog.
- void window_update_direction ()

  Function to update direction search method widgets view in the main window.
- void window_update ()

  Function to update the main window view.
- void window_set_algorithm ()

  Function to avoid memory errors changing the algorithm.
- void window_set_experiment ()

  Function to set the experiment data in the main window.

- void window_remove_experiment ()

  Function to remove an experiment in the main window.
- void window_add_experiment ()

  Function to add an experiment in the main window.
- void window_name_experiment ()

  Function to set the experiment name in the main window.
- void window_weight_experiment ()

  Function to update the experiment weight in the main window.
- void window_inputs_experiment ()

  Function to update the experiment input templates number in the main window.
- void window_template_experiment (void *data)

  Function to update the experiment i-th input template in the main window.
- void window_set_variable ()

  Function to set the variable data in the main window.
- void window_remove_variable ()

  Function to remove a variable in the main window.
- void window_add_variable ()

  Function to add a variable in the main window.
- void window_label_variable ()

  Function to set the variable label in the main window.
- void window_precision_variable ()

  Function to update the variable precision in the main window.
- void window_rangemin_variable ()

  Function to update the variable rangemin in the main window.
- void window_rangemax_variable ()

  Function to update the variable rangemax in the main window.
- void window_rangeminabs_variable ()

  Function to update the variable rangeminabs in the main window.
- void window_rangemaxabs_variable ()

  Function to update the variable rangemaxabs in the main window.
- void window_update_variable ()

  Function to update the variable data in the main window.
- int window_read (char *filename)

  Function to read the input data of a file.
- void window_open ()

  Function to open the input data.
- void window_new ()

  Function to open the main window.

Variables

- const char * logo [ ]

  Logo pixmap.
- Options options [1]

  Options struct to define the options dialog.
- Running running [1]

  Running struct to define the running dialog.
- Window window [1]

  Window struct to define the main interface window.

5.5.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.5.2 Function Documentation

5.5.2.1 unsigned int gtk_array_get_active (GtkRadioButton * *array[ ]*, unsigned int *n*)

Function to get the active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 342 of file utils.c.

```
00343 {
00344   unsigned int i;
00345   for (i = 0; i < n; ++i)
00346     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00347       break;
00348   return i;
00349 }
```

5.5.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 201 of file interface.c.

```
00202 {
00203   unsigned int i, j;
00204   char *buffer;
00205   xmlDoc *doc;
00206   xmlNode *node, *child;
00207   GFile *file, *file2;
00208
00209 #if DEBUG
00210   fprintf (stderr, "input_save: start\n");
00211 #endif
00212
00213   // Getting the input file directory
00214   input->name = g_path_get_basename (filename);
00215   input->directory = g_path_get_dirname (filename);
00216   file = g_file_new_for_path (input->directory);
```

```
00217
00218   // Opening the input file
00219   doc = xmlNewDoc ((const xmlChar *) "1.0");
00220
00221   // Setting root XML node
00222   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00223   xmlDocSetRootElement (doc, node);
00224
00225   // Adding properties to the root XML node
00226   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00227     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00228   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
00229     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
00230   file2 = g_file_new_for_path (input->simulator);
00231   buffer = g_file_get_relative_path (file, file2);
00232   g_object_unref (file2);
00233   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00234   g_free (buffer);
00235   if (input->evaluator)
00236     {
00237       file2 = g_file_new_for_path (input->evaluator);
00238       buffer = g_file_get_relative_path (file, file2);
00239       g_object_unref (file2);
00240       if (xmlStrlen ((xmlChar *) buffer))
00241         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00242       g_free (buffer);
00243     }
00244   if (input->seed != DEFAULT_RANDOM_SEED)
00245     xml_node_set_uint (node, XML_SEED, input->seed);
00246
00247   // Setting the algorithm
00248   buffer = (char *) g_malloc (64);
00249   switch (input->algorithm)
00250     {
00251     case ALGORITHM_MONTE_CARLO:
00252       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00253       snprintf (buffer, 64, "%u", input->nsimulations);
00254       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00255       snprintf (buffer, 64, "%u", input->niterations);
00256       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00257       snprintf (buffer, 64, "%.3lg", input->tolerance);
00258       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00259       snprintf (buffer, 64, "%u", input->nbest);
00260       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00261       input_save_direction (node);
00262       break;
00263     case ALGORITHM_SWEEP:
00264       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00265       snprintf (buffer, 64, "%u", input->niterations);
00266       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00267       snprintf (buffer, 64, "%.3lg", input->tolerance);
00268       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00269       snprintf (buffer, 64, "%u", input->nbest);
00270       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00271       input_save_direction (node);
00272       break;
00273     default:
00274       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00275       snprintf (buffer, 64, "%u", input->nsimulations);
00276       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00277       snprintf (buffer, 64, "%u", input->niterations);
00278       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00279       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00280       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00281       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00282       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00283       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00284       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00285       break;
00286     }
00287   g_free (buffer);
00288   if (input->thresold != 0.)
00289     xml_node_set_float (node, XML_THRESOLD, input->
      thresold);
00290
00291   // Setting the experimental data
00292   for (i = 0; i < input->nexperiments; ++i)
00293     {
00294       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00295       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
00296       if (input->weight[i] != 1.)
00297         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
00298       for (j = 0; j < input->ninputs; ++j)
00299         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
```

```
00300     }
00301
00302   // Setting the variables data
00303   for (i = 0; i < input->nvariables; ++i)
00304     {
00305       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00306       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
00307       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
00308       if (input->rangeminabs[i] != -G_MAXDOUBLE)
00309         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
00310       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
00311       if (input->rangemaxabs[i] != G_MAXDOUBLE)
00312         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
00313       if (input->precision[i] != DEFAULT_PRECISION)
00314         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
00315       if (input->algorithm == ALGORITHM_SWEEP)
00316         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
00317       else if (input->algorithm == ALGORITHM_GENETIC)
00318         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
00319       if (input->nsteps)
00320         xml_node_set_float (child, XML_STEP, input->
      step[i]);
00321     }
00322
00323   // Saving the error norm
00324   switch (input->norm)
00325     {
00326     case ERROR_NORM_MAXIMUM:
00327       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00328       break;
00329     case ERROR_NORM_P:
00330       xmlSetProp (node, XML_NORM, XML_P);
00331       xml_node_set_float (node, XML_P, input->p);
00332       break;
00333     case ERROR_NORM_TAXICAB:
00334       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00335     }
00336
00337   // Saving the XML file
00338   xmlSaveFormatFile (filename, doc, 1);
00339
00340   // Freeing memory
00341   xmlFreeDoc (doc);
00342
00343 #if DEBUG
00344   fprintf (stderr, "input_save: end\n");
00345 #endif
00346 }
```

Here is the call graph for this function:

5.5.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 450 of file interface.c.

```
00451 {
00452   unsigned int i;
00453 #if DEBUG
00454   fprintf (stderr, "window_get_algorithm: start\n");
00455 #endif
00456   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00457 #if DEBUG
00458   fprintf (stderr, "window_get_algorithm: %u\n", i);
00459   fprintf (stderr, "window_get_algorithm: end\n");
00460 #endif
00461   return i;
00462 }
```

Here is the call graph for this function:

5.5.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 470 of file interface.c.

```
00471 {
00472   unsigned int i;
00473 #if DEBUG
00474   fprintf (stderr, "window_get_direction: start\n");
00475 #endif
00476   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00477 #if DEBUG
00478   fprintf (stderr, "window_get_direction: %u\n", i);
00479   fprintf (stderr, "window_get_direction: end\n");
00480 #endif
00481   return i;
00482 }
```

Here is the call graph for this function:

5.5.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 490 of file interface.c.

```
00491 {
00492   unsigned int i;
00493 #if DEBUG
00494   fprintf (stderr, "window_get_norm: start\n");
00495 #endif
00496   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00497 #if DEBUG
00498   fprintf (stderr, "window_get_norm: %u\n", i);
00499   fprintf (stderr, "window_get_norm: end\n");
00500 #endif
00501   return i;
00502 }
```

Here is the call graph for this function:

5.5.2.6 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 1597 of file interface.c.

```
01598 {
01599   unsigned int i;
01600   char *buffer;
01601 #if DEBUG
01602   fprintf (stderr, "window_read: start\n");
01603 #endif
01604
01605   // Reading new input file
01606   input_free ();
01607   if (!input_open (filename))
01608     return 0;
01609
01610   // Setting GTK+ widgets data
01611   gtk_entry_set_text (window->entry_result, input->result);
01612   gtk_entry_set_text (window->entry_variables, input->
      variables);
01613   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01614   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01615                                  (window->button_simulator), buffer);
01616   g_free (buffer);
01617   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01618                                 (size_t) input->evaluator);
01619   if (input->evaluator)
01620     {
01621       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01622       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01623                                      (window->button_evaluator), buffer);
01624       g_free (buffer);
01625     }
01626   gtk_toggle_button_set_active
01627     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01628   switch (input->algorithm)
01629     {
01630     case ALGORITHM_MONTE_CARLO:
01631       gtk_spin_button_set_value (window->spin_simulations,
01632                                  (gdouble) input->nsimulations);
01633     case ALGORITHM_SWEEP:
01634       gtk_spin_button_set_value (window->spin_iterations,
01635                                  (gdouble) input->niterations);
01636       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01637       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01638       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
01639                                     input->nsteps);
01640       if (input->nsteps)
01641         {
01642           gtk_toggle_button_set_active
01643             (GTK_TOGGLE_BUTTON (window->button_direction
01644                                 [input->direction]), TRUE);
01645           gtk_spin_button_set_value (window->spin_steps,
01646                                      (gdouble) input->nsteps);
01647           gtk_spin_button_set_value (window->spin_relaxation,
01648                                      (gdouble) input->relaxation);
01649           switch (input->direction)
01650             {
01651             case DIRECTION_METHOD_RANDOM:
01652               gtk_spin_button_set_value (window->spin_estimates,
01653                                          (gdouble) input->nestimates);
01654             }
01655         }
01656       break;
01657     default:
01658       gtk_spin_button_set_value (window->spin_population,
01659                                  (gdouble) input->nsimulations);
01660       gtk_spin_button_set_value (window->spin_generations,
01661                                  (gdouble) input->niterations);
01662       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01663       gtk_spin_button_set_value (window->spin_reproduction,
01664                                  input->reproduction_ratio);
```

```
01665       gtk_spin_button_set_value (window->spin_adaptation,
01666                                  input->adaptation_ratio);
01667     }
01668   gtk_toggle_button_set_active
01669     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01670   gtk_spin_button_set_value (window->spin_p, input->p);
01671   gtk_spin_button_set_value (window->spin_thresold, input->
      thresold);
01672   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01673   g_signal_handler_block (window->button_experiment,
01674                           window->id_experiment_name);
01675   gtk_combo_box_text_remove_all (window->combo_experiment);
01676   for (i = 0; i < input->nexperiments; ++i)
01677     gtk_combo_box_text_append_text (window->combo_experiment,
01678                                     input->experiment[i]);
01679   g_signal_handler_unblock
01680     (window->button_experiment, window->
      id_experiment_name);
01681   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01682   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01683   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01684   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01685   gtk_combo_box_text_remove_all (window->combo_variable);
01686   for (i = 0; i < input->nvariables; ++i)
01687     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
01688   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01689   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01690   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01691   window_set_variable ();
01692   window_update ();
01693
01694 #if DEBUG
01695   fprintf (stderr, "window_read: end\n");
01696 #endif
01697   return 1;
01698 }
```

Here is the call graph for this function:

5.5.2.7 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 543 of file interface.c.

```
00544 {
00545   GtkFileChooserDialog *dlg;
00546   GtkFileFilter *filter;
00547   char *buffer;
00548
00549 #if DEBUG
00550   fprintf (stderr, "window_save: start\n");
00551 #endif
00552
00553   // Opening the saving dialog
00554   dlg = (GtkFileChooserDialog *)
00555     gtk_file_chooser_dialog_new (gettext ("Save file"),
00556                                  window->window,
00557                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00558                                  gettext ("_Cancel"),
00559                                  GTK_RESPONSE_CANCEL,
00560                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00561   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00562   buffer = g_build_filename (input->directory, input->name, NULL);
00563   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00564   g_free (buffer);
00565
00566   // Adding XML filter
```

```
00567   filter = (GtkFileFilter *) gtk_file_filter_new ();
00568   gtk_file_filter_set_name (filter, "XML");
00569   gtk_file_filter_add_pattern (filter, "*.xml");
00570   gtk_file_filter_add_pattern (filter, "*.XML");
00571   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00572
00573   // If OK response then saving
00574   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00575     {
00576
00577       // Adding properties to the root XML node
00578       input->simulator = gtk_file_chooser_get_filename
00579         (GTK_FILE_CHOOSER (window->button_simulator));
00580       if (gtk_toggle_button_get_active
00581           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00582         input->evaluator = gtk_file_chooser_get_filename
00583           (GTK_FILE_CHOOSER (window->button_evaluator));
00584       else
00585         input->evaluator = NULL;
00586       input->result
00587         = (char *) xmlStrdup ((const xmlChar *)
00588                               gtk_entry_get_text (window->entry_result));
00589       input->variables
00590         = (char *) xmlStrdup ((const xmlChar *)
00591                               gtk_entry_get_text (window->entry_variables));
00592
00593       // Setting the algorithm
00594       switch (window_get_algorithm ())
00595         {
00596         case ALGORITHM_MONTE_CARLO:
00597           input->algorithm = ALGORITHM_MONTE_CARLO;
00598           input->nsimulations
00599             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00600           input->niterations
00601             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00602           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00603           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00604           window_save_direction ();
00605           break;
00606         case ALGORITHM_SWEEP:
00607           input->algorithm = ALGORITHM_SWEEP;
00608           input->niterations
00609             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00610           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00611           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00612           window_save_direction ();
00613           break;
00614         default:
00615           input->algorithm = ALGORITHM_GENETIC;
00616           input->nsimulations
00617             = gtk_spin_button_get_value_as_int (window->spin_population);
00618           input->niterations
00619             = gtk_spin_button_get_value_as_int (window->spin_generations);
00620           input->mutation_ratio
00621             = gtk_spin_button_get_value (window->spin_mutation);
00622           input->reproduction_ratio
00623             = gtk_spin_button_get_value (window->spin_reproduction);
00624           input->adaptation_ratio
00625             = gtk_spin_button_get_value (window->spin_adaptation);
00626           break;
00627         }
00628       input->norm = window_get_norm ();
00629       input->p = gtk_spin_button_get_value (window->spin_p);
00630       input->thresold = gtk_spin_button_get_value (window->
      spin_thresold);
00631
00632       // Saving the XML file
00633       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00634       input_save (buffer);
00635
00636       // Closing and freeing memory
00637       g_free (buffer);
00638       gtk_widget_destroy (GTK_WIDGET (dlg));
00639 #if DEBUG
00640       fprintf (stderr, "window_save: end\n");
00641 #endif
00642       return 1;
00643     }
00644
00645   // Closing and freeing memory
00646   gtk_widget_destroy (GTK_WIDGET (dlg));
00647 #if DEBUG
00648   fprintf (stderr, "window_save: end\n");
```

```
00649 #endif
00650   return 0;
00651 }
```

Here is the call graph for this function:

5.5.2.8 void window_template_experiment (void * *data*)

Function to update the experiment i-th input template in the main window.

Parameters

data	Callback data (i-th input template).

Definition at line 1201 of file interface.c.

```
01202 {
01203   unsigned int i, j;
01204   char *buffer;
01205   GFile *file1, *file2;
01206 #if DEBUG
01207   fprintf (stderr, "window_template_experiment: start\n");
01208 #endif
01209   i = (size_t) data;
01210   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01211   file1
01212     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01213   file2 = g_file_new_for_path (input->directory);
01214   buffer = g_file_get_relative_path (file2, file1);
01215   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
01216   g_free (buffer);
01217   g_object_unref (file2);
01218   g_object_unref (file1);
01219 #if DEBUG
01220   fprintf (stderr, "window_template_experiment: end\n");
01221 #endif
01222 }
```

5.6 interface.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
```

```
00049 {
00050   char *template[MAX_NINPUTS];
00051   char *name;
00052   double weight;
00054 } Experiment;
00055
00060 typedef struct
00061 {
00062   char *label;
00063   double rangemin;
00064   double rangemax;
00065   double rangeminabs;
00066   double rangemaxabs;
00067   double step;
00069   unsigned int precision;
00070   unsigned int nsweeps;
00071   unsigned int nbits;
00072 } Variable;
00073
00078 typedef struct
00079 {
00080   GtkDialog *dialog;
00081   GtkGrid *grid;
00082   GtkLabel *label_seed;
00084   GtkSpinButton *spin_seed;
00086   GtkLabel *label_threads;
00087   GtkSpinButton *spin_threads;
00088   GtkLabel *label_direction;
00089   GtkSpinButton *spin_direction;
00090 } Options;
00091
00096 typedef struct
00097 {
00098   GtkDialog *dialog;
00099   GtkLabel *label;
00100 } Running;
00101
00106 typedef struct
00107 {
00108   GtkWindow *window;
00109   GtkGrid *grid;
00110   GtkToolbar *bar_buttons;
00111   GtkToolButton *button_open;
00112   GtkToolButton *button_save;
00113   GtkToolButton *button_run;
00114   GtkToolButton *button_options;
00115   GtkToolButton *button_help;
00116   GtkToolButton *button_about;
00117   GtkToolButton *button_exit;
00118   GtkGrid *grid_files;
00119   GtkLabel *label_simulator;
00120   GtkFileChooserButton *button_simulator;
00122   GtkCheckButton *check_evaluator;
00123   GtkFileChooserButton *button_evaluator;
00125   GtkLabel *label_result;
00126   GtkEntry *entry_result;
00127   GtkLabel *label_variables;
00128   GtkEntry *entry_variables;
00129   GtkFrame *frame_norm;
00130   GtkGrid *grid_norm;
00131   GtkRadioButton *button_norm[NNORMS];
00133   GtkLabel *label_p;
00134   GtkSpinButton *spin_p;
00135   GtkScrolledWindow *scrolled_p;
00137   GtkFrame *frame_algorithm;
00138   GtkGrid *grid_algorithm;
00139   GtkRadioButton *button_algorithm[NALGORITHMS];
00141   GtkLabel *label_simulations;
00142   GtkSpinButton *spin_simulations;
00144   GtkLabel *label_iterations;
00145   GtkSpinButton *spin_iterations;
00147   GtkLabel *label_tolerance;
00148   GtkSpinButton *spin_tolerance;
00149   GtkLabel *label_bests;
00150   GtkSpinButton *spin_bests;
00151   GtkLabel *label_population;
00152   GtkSpinButton *spin_population;
00154   GtkLabel *label_generations;
00155   GtkSpinButton *spin_generations;
00157   GtkLabel *label_mutation;
00158   GtkSpinButton *spin_mutation;
00159   GtkLabel *label_reproduction;
00160   GtkSpinButton *spin_reproduction;
00162   GtkLabel *label_adaptation;
00163   GtkSpinButton *spin_adaptation;
00165   GtkCheckButton *check_direction;
00167   GtkGrid *grid_direction;
```

```
00169   GtkRadioButton *button_direction[NDIRECTIONS];
00171   GtkLabel *label_steps;
00172   GtkSpinButton *spin_steps;
00173   GtkLabel *label_estimates;
00174   GtkSpinButton *spin_estimates;
00176   GtkLabel *label_relaxation;
00178   GtkSpinButton *spin_relaxation;
00180   GtkLabel *label_thresold;
00181   GtkSpinButton *spin_thresold;
00182   GtkScrolledWindow *scrolled_thresold;
00184   GtkFrame *frame_variable;
00185   GtkGrid *grid_variable;
00186   GtkComboBoxText *combo_variable;
00188   GtkButton *button_add_variable;
00189   GtkButton *button_remove_variable;
00190   GtkLabel *label_variable;
00191   GtkEntry *entry_variable;
00192   GtkLabel *label_min;
00193   GtkSpinButton *spin_min;
00194   GtkScrolledWindow *scrolled_min;
00195   GtkLabel *label_max;
00196   GtkSpinButton *spin_max;
00197   GtkScrolledWindow *scrolled_max;
00198   GtkCheckButton *check_minabs;
00199   GtkSpinButton *spin_minabs;
00200   GtkScrolledWindow *scrolled_minabs;
00201   GtkCheckButton *check_maxabs;
00202   GtkSpinButton *spin_maxabs;
00203   GtkScrolledWindow *scrolled_maxabs;
00204   GtkLabel *label_precision;
00205   GtkSpinButton *spin_precision;
00206   GtkLabel *label_sweeps;
00207   GtkSpinButton *spin_sweeps;
00208   GtkLabel *label_bits;
00209   GtkSpinButton *spin_bits;
00210   GtkLabel *label_step;
00211   GtkSpinButton *spin_step;
00212   GtkScrolledWindow *scrolled_step;
00213   GtkFrame *frame_experiment;
00214   GtkGrid *grid_experiment;
00215   GtkComboBoxText *combo_experiment;
00216   GtkButton *button_add_experiment;
00217   GtkButton *button_remove_experiment;
00218   GtkLabel *label_experiment;
00219   GtkFileChooserButton *button_experiment;
00221   GtkLabel *label_weight;
00222   GtkSpinButton *spin_weight;
00223   GtkCheckButton *check_template[MAX_NINPUTS];
00225   GtkFileChooserButton *button_template[MAX_NINPUTS];
00227   GdkPixbuf *logo;
00228   Experiment *experiment;
00229   Variable *variable;
00230   char *application_directory;
00231   gulong id_experiment;
00232   gulong id_experiment_name;
00233   gulong id_variable;
00234   gulong id_variable_label;
00235   gulong id_template[MAX_NINPUTS];
00237   gulong id_input[MAX_NINPUTS];
00239   unsigned int nexperiments;
00240   unsigned int nvariables;
00241 } Window;
00242
00243 // Global variables
00244 extern const char *logo[];
00245 extern Options options[1];
00246 extern Running running[1];
00247 extern Window window[1];
00248
00249 // Public functions
00250 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00251 void input_save (char *filename);
00252 void options_new ();
00253 void running_new ();
00254 unsigned int window_get_algorithm ();
00255 unsigned int window_get_direction ();
00256 unsigned int window_get_norm ();
00257 void window_save_direction ();
00258 int window_save ();
00259 void window_run ();
00260 void window_help ();
00261 void window_update_direction ();
00262 void window_update ();
00263 void window_set_algorithm ();
00264 void window_set_experiment ();
00265 void window_remove_experiment ();
00266 void window_add_experiment ();
```

```
00267 void window_name_experiment ();
00268 void window_weight_experiment ();
00269 void window_inputs_experiment ();
00270 void window_template_experiment (void *data);
00271 void window_set_variable ();
00272 void window_remove_variable ();
00273 void window_add_variable ();
00274 void window_label_variable ();
00275 void window_precision_variable ();
00276 void window_rangemin_variable ();
00277 void window_rangemax_variable ();
00278 void window_rangeminabs_variable ();
00279 void window_rangemaxabs_variable ();
00280 void window_update_variable ();
00281 int window_read (char *filename);
00282 void window_open ();
00283 void window_new ();
00284
00285 #endif
```

5.7 main.c File Reference

Main source file.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <alloca.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"
#include "interface.h"
```

Include dependency graph for main.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG 0

  Macro to debug.

Functions

- int main (int argn, char **argc)

  Main function.

5.7.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file main.c.

5.7.2 Function Documentation

5.7.2.1 int main (int *argn*, char ** *argc*)

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

0 on success, >0 on error.

Definition at line 81 of file main.c.

```
00082 {
00083 #if HAVE_GTK
00084   char *buffer;
00085 #endif
00086
00087   // Starting pseudo-random numbers generator
00088   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00089
00090   // Allowing spaces in the XML data file
00091   xmlKeepBlanksDefault (0);
00092
00093   // Starting MPI
00094 #if HAVE_MPI
00095   MPI_Init (&argn, &argc);
00096   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00097   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00098   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00099 #else
00100   ntasks = 1;
00101 #endif
00102
00103   // Resetting result and variables file names
00104   input->result = input->variables = NULL;
00105
00106 #if HAVE_GTK
00107
00108   // Getting threads number and pseudo-random numbers generator seed
00109   nthreads_direction = nthreads = cores_number ();
00110   optimize->seed = DEFAULT_RANDOM_SEED;
00111
00112   // Setting local language and international floating point numbers notation
00113   setlocale (LC_ALL, "");
00114   setlocale (LC_NUMERIC, "C");
00115   window->application_directory = g_get_current_dir ();
00116   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
00117   bindtextdomain (PROGRAM_INTERFACE, buffer);
00118   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00119   textdomain (PROGRAM_INTERFACE);
00120
00121   // Initing GTK+
00122   gtk_disable_setlocale ();
00123   gtk_init (&argn, &argc);
00124
00125   // Opening the main window
00126   window_new ();
00127   gtk_main ();
```

```
00128
00129   // Freeing memory
00130   input_free ();
00131   g_free (buffer);
00132   gtk_widget_destroy (GTK_WIDGET (window->window));
00133   g_free (window->application_directory);
00134
00135 #else
00136
00137   // Checking syntax
00138   if (argn < 2)
00139     {
00140       printf ("The syntax is:\n"
00141               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00142               "[variables_file]\n");
00143       return 1;
00144     }
00145
00146   // Getting threads number and pseudo-random numbers generator seed
00147   nthreads_direction = nthreads = cores_number ();
00148   optimize->seed = DEFAULT_RANDOM_SEED;
00149   if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00150     {
00151       nthreads_direction = nthreads = atoi (argc[2]);
00152       if (!nthreads)
00153         {
00154           printf ("Bad threads number\n");
00155           return 2;
00156         }
00157       argc += 2;
00158       argn -= 2;
00159       if (argn > 2 && !strcmp (argc[1], "-seed"))
00160         {
00161           optimize->seed = atoi (argc[2]);
00162           argc += 2;
00163           argn -= 2;
00164         }
00165     }
00166   else if (argn > 2 && !strcmp (argc[1], "-seed"))
00167     {
00168       optimize->seed = atoi (argc[2]);
00169       argc += 2;
00170       argn -= 2;
00171       if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00172         {
00173           nthreads_direction = nthreads = atoi (argc[2]);
00174           if (!nthreads)
00175             {
00176               printf ("Bad threads number\n");
00177               return 2;
00178             }
00179           argc += 2;
00180           argn -= 2;
00181         }
00182     }
00183   printf ("nthreads=%u\n", nthreads);
00184   printf ("seed=%lu\n", optimize->seed);
00185
00186   // Checking arguments
00187   if (argn > 4 || argn < 2)
00188     {
00189       printf ("The syntax is:\n"
00190               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00191               "[variables_file]\n");
00192       return 1;
00193     }
00194   if (argn > 2)
00195     input->result = argc[2];
00196   if (argn == 4)
00197     input->variables = argc[3];
00198
00199   // Making optimization
00200   if (input_open (argc[1]))
00201     optimize_open ();
00202
00203   // Freeing memory
00204   optimize_free ();
00205
00206 #endif
00207
00208   // Closing MPI
00209 #if HAVE_MPI
00210   MPI_Finalize ();
00211 #endif
00212
00213   // Freeing memory
00214   gsl_rng_free (optimize->rng);
```

```
00215
00216   // Closing
00217   return 0;
00218 }
```

Here is the call graph for this function:

5.8 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <glib/gstdio.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #elif !defined (BSD)
00053 #include <alloca.h>
00054 #endif
00055 #if HAVE_MPI
00056 #include <mpi.h>
00057 #endif
00058 #if HAVE_GTK
00059 #include <gio/gio.h>
00060 #include <gtk/gtk.h>
00061 #endif
00062 #include "genetic/genetic.h"
00063 #include "utils.h"
00064 #include "optimize.h"
00065 #if HAVE_GTK
00066 #include "interface.h"
00067 #endif
00068
00069 #define DEBUG 0
00070
00071
00080 int
00081 main (int argn, char **argc)
00082 {
00083 #if HAVE_GTK
00084   char *buffer;
00085 #endif
00086
00087   // Starting pseudo-random numbers generator
```

```
00088   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00089
00090   // Allowing spaces in the XML data file
00091   xmlKeepBlanksDefault (0);
00092
00093   // Starting MPI
00094 #if HAVE_MPI
00095   MPI_Init (&argn, &argc);
00096   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00097   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00098   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00099 #else
00100   ntasks = 1;
00101 #endif
00102
00103   // Resetting result and variables file names
00104   input->result = input->variables = NULL;
00105
00106 #if HAVE_GTK
00107
00108   // Getting threads number and pseudo-random numbers generator seed
00109   nthreads_direction = nthreads = cores_number ();
00110   optimize->seed = DEFAULT_RANDOM_SEED;
00111
00112   // Setting local language and international floating point numbers notation
00113   setlocale (LC_ALL, "");
00114   setlocale (LC_NUMERIC, "C");
00115   window->application_directory = g_get_current_dir ();
00116   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
00117   bindtextdomain (PROGRAM_INTERFACE, buffer);
00118   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00119   textdomain (PROGRAM_INTERFACE);
00120
00121   // Initing GTK+
00122   gtk_disable_setlocale ();
00123   gtk_init (&argn, &argc);
00124
00125   // Opening the main window
00126   window_new ();
00127   gtk_main ();
00128
00129   // Freeing memory
00130   input_free ();
00131   g_free (buffer);
00132   gtk_widget_destroy (GTK_WIDGET (window->window));
00133   g_free (window->application_directory);
00134
00135 #else
00136
00137   // Checking syntax
00138   if (argn < 2)
00139     {
00140       printf ("The syntax is:\n"
00141               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00142               "[variables_file]\n");
00143       return 1;
00144     }
00145
00146   // Getting threads number and pseudo-random numbers generator seed
00147   nthreads_direction = nthreads = cores_number ();
00148   optimize->seed = DEFAULT_RANDOM_SEED;
00149   if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00150     {
00151       nthreads_direction = nthreads = atoi (argc[2]);
00152       if (!nthreads)
00153         {
00154           printf ("Bad threads number\n");
00155           return 2;
00156         }
00157       argc += 2;
00158       argn -= 2;
00159       if (argn > 2 && !strcmp (argc[1], "-seed"))
00160         {
00161           optimize->seed = atoi (argc[2]);
00162           argc += 2;
00163           argn -= 2;
00164         }
00165     }
00166   else if (argn > 2 && !strcmp (argc[1], "-seed"))
00167     {
00168       optimize->seed = atoi (argc[2]);
00169       argc += 2;
00170       argn -= 2;
00171       if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00172         {
00173           nthreads_direction = nthreads = atoi (argc[2]);
```

```
00174           if (!nthreads)
00175             {
00176               printf ("Bad threads number\n");
00177               return 2;
00178             }
00179           argc += 2;
00180           argn -= 2;
00181         }
00182     }
00183   printf ("nthreads=%u\n", nthreads);
00184   printf ("seed=%lu\n", optimize->seed);
00185
00186   // Checking arguments
00187   if (argn > 4 || argn < 2)
00188     {
00189       printf ("The syntax is:\n"
00190               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00191               "[variables_file]\n");
00192       return 1;
00193     }
00194   if (argn > 2)
00195     input->result = argc[2];
00196   if (argn == 4)
00197     input->variables = argc[3];
00198
00199   // Making optimization
00200   if (input_open (argc[1]))
00201     optimize_open ();
00202
00203   // Freeing memory
00204   optimize_free ();
00205
00206 #endif
00207
00208   // Closing MPI
00209 #if HAVE_MPI
00210   MPI_Finalize ();
00211 #endif
00212
00213   // Freeing memory
00214   gsl_rng_free (optimize->rng);
00215
00216   // Closing
00217   return 0;
00218 }
```

5.9 optimize.c File Reference

Source file to define the optimization functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"
```

Include dependency graph for optimize.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG 0

Macro to debug.

- #define RM "rm"

  Macro to define the shell remove command.

Functions

- void input_new ()

  Function to create a new Input struct.

- void input_free ()

  Function to free the memory of the input file data.

- int input_open (char *filename)

  Function to open the input file.

- void optimize_input (unsigned int simulation, char *input, GMappedFile *template)

  Function to write the simulation input file.

- double optimize_parse (unsigned int simulation, unsigned int experiment)

  Function to parse input files, simulating and calculating the \ objective function.

- double optimize_norm_euclidian (unsigned int simulation)

  Function to calculate the Euclidian error norm.

- double optimize_norm_maximum (unsigned int simulation)

  Function to calculate the maximum error norm.

- double optimize_norm_p (unsigned int simulation)

  Function to calculate the P error norm.

- double optimize_norm_taxicab (unsigned int simulation)

  Function to calculate the taxicab error norm.

- void optimize_print ()

  Function to print the results.

- void optimize_save_variables (unsigned int simulation, double error)

  Function to save in a file the variables and the error.

- void optimize_best (unsigned int simulation, double value)

  Function to save the best simulations.

- void optimize_sequential ()

  Function to optimize sequentially.

- void * optimize_thread (ParallelData *data)

  Function to optimize on a thread.

- void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 optimization results.

- void optimize_synchronise ()

  Function to synchronise the optimization results of MPI tasks.

- void optimize_sweep ()

  Function to optimize with the sweep algorithm.

- void optimize_MonteCarlo ()

  Function to optimize with the Monte-Carlo algorithm.

- void optimize_best_direction (unsigned int simulation, double value)

  Function to save the best simulation in a direction search method.

- void optimize_direction_sequential (unsigned int simulation)

  Function to estimate the direction search sequentially.

- void * optimize_direction_thread (ParallelData *data)

  Function to estimate the direction search on a thread.

- double optimize_estimate_direction_random (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.

- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- void optimize_step_direction (unsigned int simulation)

  Function to do a step of the direction search method.
- void optimize_direction ()

  Function to optimize with a direction search method.
- double optimize_genetic_objective (Entity *entity)

  Function to calculate the objective function of an entity.
- void optimize_genetic ()

  Function to optimize with the genetic algorithm.
- void optimize_save_old ()

  Function to save the best results on iterative methods.
- void optimize_merge_old ()

  Function to merge the best results with the previous step best results on iterative methods.
- void optimize_refine ()

  Function to refine the search ranges of the variables in iterative algorithms.
- void optimize_step ()

  Function to do a step of the iterative algorithm.
- void optimize_iterate ()

  Function to iterate the algorithm.
- void optimize_free ()

  Function to free the memory used by the Optimize struct.
- void optimize_open ()

  Function to open and perform a optimization.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- unsigned int nthreads_direction

  Number of threads for the direction search method.
- GMutex mutex [1]

  Mutex struct.
- void(* optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(* optimize_estimate_direction)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the direction.
- double(* optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Input input [1]

  Input struct to define the input file to mpcotool.
- Optimize optimize [1]

  Optimization data.
- const xmlChar * result_name = (xmlChar *) "result"

  Name of the result file.
- const xmlChar * variables_name = (xmlChar *) "variables"

  Name of the variables file.
- const xmlChar * template [MAX_NINPUTS]

Array of xmlChar strings with template labels.

- const char * format [NPRECISIONS]

  Array of C-strings with variable formats.

- const double precision [NPRECISIONS]

  Array of variable precisions.

5.9.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file optimize.c.

5.9.2 Function Documentation

5.9.2.1 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 188 of file optimize.c.

```
00189 {
00190   char buffer2[64];
00191   char *buffert[MAX_NINPUTS] =
00192     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00193   xmlDoc *doc;
00194   xmlNode *node, *child;
00195   xmlChar *buffer;
00196   char *msg;
00197   int error_code;
00198   unsigned int i;
00199
00200 #if DEBUG
00201   fprintf (stderr, "input_open: start\n");
00202 #endif
00203
00204   // Resetting input data
00205   buffer = NULL;
00206   input_new ();
00207
00208   // Parsing the input file
00209 #if DEBUG
00210   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00211 #endif
00212   doc = xmlParseFile (filename);
00213   if (!doc)
00214     {
00215       msg = gettext ("Unable to parse the input file");
00216       goto exit_on_error;
00217     }
00218
```

```
00219   // Getting the root node
00220 #if DEBUG
00221   fprintf (stderr, "input_open: getting the root node\n");
00222 #endif
00223   node = xmlDocGetRootElement (doc);
00224   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00225     {
00226       msg = gettext ("Bad root XML node");
00227       goto exit_on_error;
00228     }
00229
00230   // Getting result and variables file names
00231   if (!input->result)
00232     {
00233       input->result = (char *) xmlGetProp (node, XML_RESULT);
00234       if (!input->result)
00235         input->result = (char *) xmlStrdup (result_name);
00236     }
00237   if (!input->variables)
00238     {
00239       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00240       if (!input->variables)
00241         input->variables = (char *) xmlStrdup (variables_name);
00242     }
00243
00244   // Opening simulator program name
00245   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00246   if (!input->simulator)
00247     {
00248       msg = gettext ("Bad simulator program");
00249       goto exit_on_error;
00250     }
00251
00252   // Opening evaluator program name
00253   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00254
00255   // Obtaining pseudo-random numbers generator seed
00256   input->seed
00257     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00258                                       &error_code);
00259   if (error_code)
00260     {
00261       msg = gettext ("Bad pseudo-random numbers generator seed");
00262       goto exit_on_error;
00263     }
00264
00265   // Opening algorithm
00266   buffer = xmlGetProp (node, XML_ALGORITHM);
00267   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00268     {
00269       input->algorithm = ALGORITHM_MONTE_CARLO;
00270
00271       // Obtaining simulations number
00272       input->nsimulations
00273         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00274       if (error_code)
00275         {
00276           msg = gettext ("Bad simulations number");
00277           goto exit_on_error;
00278         }
00279     }
00280   else if (!xmlStrcmp (buffer, XML_SWEEP))
00281     input->algorithm = ALGORITHM_SWEEP;
00282   else if (!xmlStrcmp (buffer, XML_GENETIC))
00283     {
00284       input->algorithm = ALGORITHM_GENETIC;
00285
00286       // Obtaining population
00287       if (xmlHasProp (node, XML_NPOPULATION))
00288         {
00289           input->nsimulations
00290             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00291           if (error_code || input->nsimulations < 3)
00292             {
00293               msg = gettext ("Invalid population number");
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           msg = gettext ("No population number");
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining generations
00304       if (xmlHasProp (node, XML_NGENERATIONS))
```

```
00305         {
00306           input->niterations
00307             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00308           if (error_code || !input->niterations)
00309             {
00310               msg = gettext ("Invalid generations number");
00311               goto exit_on_error;
00312             }
00313         }
00314       else
00315         {
00316           msg = gettext ("No generations number");
00317           goto exit_on_error;
00318         }
00319
00320       // Obtaining mutation probability
00321       if (xmlHasProp (node, XML_MUTATION))
00322         {
00323           input->mutation_ratio
00324             = xml_node_get_float (node, XML_MUTATION, &error_code);
00325           if (error_code || input->mutation_ratio < 0.
00326               || input->mutation_ratio >= 1.)
00327             {
00328               msg = gettext ("Invalid mutation probability");
00329               goto exit_on_error;
00330             }
00331         }
00332       else
00333         {
00334           msg = gettext ("No mutation probability");
00335           goto exit_on_error;
00336         }
00337
00338       // Obtaining reproduction probability
00339       if (xmlHasProp (node, XML_REPRODUCTION))
00340         {
00341           input->reproduction_ratio
00342             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00343           if (error_code || input->reproduction_ratio < 0.
00344               || input->reproduction_ratio >= 1.0)
00345             {
00346               msg = gettext ("Invalid reproduction probability");
00347               goto exit_on_error;
00348             }
00349         }
00350       else
00351         {
00352           msg = gettext ("No reproduction probability");
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining adaptation probability
00357       if (xmlHasProp (node, XML_ADAPTATION))
00358         {
00359           input->adaptation_ratio
00360             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00361           if (error_code || input->adaptation_ratio < 0.
00362               || input->adaptation_ratio >= 1.)
00363             {
00364               msg = gettext ("Invalid adaptation probability");
00365               goto exit_on_error;
00366             }
00367         }
00368       else
00369         {
00370           msg = gettext ("No adaptation probability");
00371           goto exit_on_error;
00372         }
00373
00374       // Checking survivals
00375       i = input->mutation_ratio * input->nsimulations;
00376       i += input->reproduction_ratio * input->
     nsimulations;
00377       i += input->adaptation_ratio * input->
     nsimulations;
00378       if (i > input->nsimulations - 2)
00379         {
00380           msg = gettext
00381             ("No enough survival entities to reproduce the population");
00382           goto exit_on_error;
00383         }
00384     }
00385   else
00386     {
00387       msg = gettext ("Unknown algorithm");
00388       goto exit_on_error;
00389     }
```

```
00390   xmlFree (buffer);
00391   buffer = NULL;
00392
00393   if (input->algorithm == ALGORITHM_MONTE_CARLO
00394       || input->algorithm == ALGORITHM_SWEEP)
00395     {
00396
00397       // Obtaining iterations number
00398       input->niterations
00399         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00400       if (error_code == 1)
00401         input->niterations = 1;
00402       else if (error_code)
00403         {
00404           msg = gettext ("Bad iterations number");
00405           goto exit_on_error;
00406         }
00407
00408       // Obtaining best number
00409       input->nbest
00410         = xml_node_get_uint_with_default (node,
       XML_NBEST, 1, &error_code);
00411       if (error_code || !input->nbest)
00412         {
00413           msg = gettext ("Invalid best number");
00414           goto exit_on_error;
00415         }
00416
00417       // Obtaining tolerance
00418       input->tolerance
00419         = xml_node_get_float_with_default (node,
       XML_TOLERANCE, 0.,
00420                                            &error_code);
00421       if (error_code || input->tolerance < 0.)
00422         {
00423           msg = gettext ("Invalid tolerance");
00424           goto exit_on_error;
00425         }
00426
00427       // Getting direction search method parameters
00428       if (xmlHasProp (node, XML_NSTEPS))
00429         {
00430           input->nsteps = xml_node_get_uint (node,
       XML_NSTEPS, &error_code);
00431           if (error_code || !input->nsteps)
00432             {
00433               msg = gettext ("Invalid steps number");
00434               goto exit_on_error;
00435             }
00436           buffer = xmlGetProp (node, XML_DIRECTION);
00437           if (!xmlStrcmp (buffer, XML_COORDINATES))
00438             input->direction = DIRECTION_METHOD_COORDINATES;
00439           else if (!xmlStrcmp (buffer, XML_RANDOM))
00440             {
00441               input->direction = DIRECTION_METHOD_RANDOM;
00442               input->nestimates
00443                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00444               if (error_code || !input->nestimates)
00445                 {
00446                   msg = gettext ("Invalid estimates number");
00447                   goto exit_on_error;
00448                 }
00449             }
00450           else
00451             {
00452               msg = gettext ("Unknown method to estimate the direction search");
00453               goto exit_on_error;
00454             }
00455           xmlFree (buffer);
00456           buffer = NULL;
00457           input->relaxation
00458             = xml_node_get_float_with_default (node,
       XML_RELAXATION,
00459                                                DEFAULT_RELAXATION, &error_code);
00460           if (error_code || input->relaxation < 0. || input->
       relaxation > 2.)
00461             {
00462               msg = gettext ("Invalid relaxation parameter");
00463               goto exit_on_error;
00464             }
00465         }
00466       else
00467         input->nsteps = 0;
00468     }
00469   // Obtaining the thresold
00470   input->thresold = xml_node_get_float_with_default (node,
       XML_THRESOLD, 0.,
```

```
00471                                                 &error_code);
00472   if (error_code)
00473     {
00474       msg = gettext ("Invalid thresold");
00475       goto exit_on_error;
00476     }
00477
00478   // Reading the experimental data
00479   for (child = node->children; child; child = child->next)
00480     {
00481       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00482         break;
00483 #if DEBUG
00484       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00485 #endif
00486       if (xmlHasProp (child, XML_NAME))
00487         buffer = xmlGetProp (child, XML_NAME);
00488       else
00489         {
00490           snprintf (buffer2, 64, "%s %u: %s",
00491                     gettext ("Experiment"),
00492                     input->nexperiments + 1, gettext ("no data file name"));
00493           msg = buffer2;
00494           goto exit_on_error;
00495         }
00496 #if DEBUG
00497       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00498 #endif
00499       input->weight = g_realloc (input->weight,
00500                                  (1 + input->nexperiments) * sizeof (double));
00501       input->weight[input->nexperiments]
00502         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00503       if (error_code)
00504         {
00505           snprintf (buffer2, 64, "%s %s: %s",
00506                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00507           msg = buffer2;
00508           goto exit_on_error;
00509         }
00510 #if DEBUG
00511       fprintf (stderr, "input_open: weight=%lg\n",
00512                input->weight[input->nexperiments]);
00513 #endif
00514       if (!input->nexperiments)
00515         input->ninputs = 0;
00516 #if DEBUG
00517       fprintf (stderr, "input_open: template[0]\n");
00518 #endif
00519       if (xmlHasProp (child, XML_TEMPLATE1))
00520         {
00521           input->template[0]
00522             = (char **) g_realloc (input->template[0],
00523                                    (1 + input->nexperiments) * sizeof (char *));
00524           buffert[0] = (char *) xmlGetProp (child, template[0]);
00525 #if DEBUG
00526           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00527                    input->nexperiments, buffert[0]);
00528 #endif
00529           if (!input->nexperiments)
00530             ++input->ninputs;
00531 #if DEBUG
00532           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00533 #endif
00534         }
00535       else
00536         {
00537           snprintf (buffer2, 64, "%s %s: %s",
00538                     gettext ("Experiment"), buffer, gettext ("no template"));
00539           msg = buffer2;
00540           goto exit_on_error;
00541         }
00542       for (i = 1; i < MAX_NINPUTS; ++i)
00543         {
00544 #if DEBUG
00545           fprintf (stderr, "input_open: template%u\n", i + 1);
00546 #endif
00547           if (xmlHasProp (child, template[i]))
00548             {
00549               if (input->nexperiments && input->ninputs <= i)
00550                 {
00551                   snprintf (buffer2, 64, "%s %s: %s",
00552                             gettext ("Experiment"),
00553                             buffer, gettext ("bad templates number"));
00554                   msg = buffer2;
00555                   while (i-- > 0)
00556                     xmlFree (buffert[i]);
```

```
00557                   goto exit_on_error;
00558                 }
00559               input->template[i] = (char **)
00560                 g_realloc (input->template[i],
00561                            (1 + input->nexperiments) * sizeof (char *));
00562               buffert[i] = (char *) xmlGetProp (child, template[i]);
00563 #if DEBUG
00564               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00565                        input->nexperiments, i + 1,
00566                        input->template[i][input->nexperiments]);
00567 #endif
00568               if (!input->nexperiments)
00569                 ++input->ninputs;
00570 #if DEBUG
00571               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00572 #endif
00573             }
00574           else if (input->nexperiments && input->ninputs > i)
00575             {
00576               snprintf (buffer2, 64, "%s %s: %s%u",
00577                         gettext ("Experiment"),
00578                         buffer, gettext ("no template"), i + 1);
00579               msg = buffer2;
00580               while (i-- > 0)
00581                 xmlFree (buffert[i]);
00582               goto exit_on_error;
00583             }
00584           else
00585             break;
00586         }
00587       input->experiment
00588         = g_realloc (input->experiment,
00589                      (1 + input->nexperiments) * sizeof (char *));
00590       input->experiment[input->nexperiments] = (char *) buffer;
00591       for (i = 0; i < input->ninputs; ++i)
00592         input->template[i][input->nexperiments] = buffert[i];
00593       ++input->nexperiments;
00594 #if DEBUG
00595       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00596 #endif
00597     }
00598   if (!input->nexperiments)
00599     {
00600       msg = gettext ("No optimization experiments");
00601       goto exit_on_error;
00602     }
00603   buffer = NULL;
00604
00605   // Reading the variables data
00606   for (; child; child = child->next)
00607     {
00608       if (xmlStrcmp (child->name, XML_VARIABLE))
00609         {
00610           snprintf (buffer2, 64, "%s %u: %s",
00611                     gettext ("Variable"),
00612                     input->nvariables + 1, gettext ("bad XML node"));
00613           msg = buffer2;
00614           goto exit_on_error;
00615         }
00616       if (xmlHasProp (child, XML_NAME))
00617         buffer = xmlGetProp (child, XML_NAME);
00618       else
00619         {
00620           snprintf (buffer2, 64, "%s %u: %s",
00621                     gettext ("Variable"),
00622                     input->nvariables + 1, gettext ("no name"));
00623           msg = buffer2;
00624           goto exit_on_error;
00625         }
00626       if (xmlHasProp (child, XML_MINIMUM))
00627         {
00628           input->rangemin = g_realloc
00629             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00630           input->rangeminabs = g_realloc
00631             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00632           input->rangemin[input->nvariables]
00633             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00634           if (error_code)
00635             {
00636               snprintf (buffer2, 64, "%s %s: %s",
00637                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00638               msg = buffer2;
00639               goto exit_on_error;
00640             }
00641           input->rangeminabs[input->nvariables]
00642             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MINIMUM,
```

```
00643                                                 -G_MAXDOUBLE, &error_code);
00644           if (error_code)
00645             {
00646               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00647                         gettext ("bad absolute minimum"));
00648               msg = buffer2;
00649               goto exit_on_error;
00650             }
00651           if (input->rangemin[input->nvariables]
00652               < input->rangeminabs[input->nvariables])
00653             {
00654               snprintf (buffer2, 64, "%s %s: %s",
00655                         gettext ("Variable"),
00656                         buffer, gettext ("minimum range not allowed"));
00657               msg = buffer2;
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           snprintf (buffer2, 64, "%s %s: %s",
00664                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00665           msg = buffer2;
00666           goto exit_on_error;
00667         }
00668       if (xmlHasProp (child, XML_MAXIMUM))
00669         {
00670           input->rangemax = g_realloc
00671             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00672           input->rangemaxabs = g_realloc
00673             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00674           input->rangemax[input->nvariables]
00675             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00676           if (error_code)
00677             {
00678               snprintf (buffer2, 64, "%s %s: %s",
00679                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00680               msg = buffer2;
00681               goto exit_on_error;
00682             }
00683           input->rangemaxabs[input->nvariables]
00684             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MAXIMUM,
00685                                                G_MAXDOUBLE, &error_code);
00686           if (error_code)
00687             {
00688               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00689                         gettext ("bad absolute maximum"));
00690               msg = buffer2;
00691               goto exit_on_error;
00692             }
00693           if (input->rangemax[input->nvariables]
00694               > input->rangemaxabs[input->nvariables])
00695             {
00696               snprintf (buffer2, 64, "%s %s: %s",
00697                         gettext ("Variable"),
00698                         buffer, gettext ("maximum range not allowed"));
00699               msg = buffer2;
00700               goto exit_on_error;
00701             }
00702         }
00703       else
00704         {
00705           snprintf (buffer2, 64, "%s %s: %s",
00706                     gettext ("Variable"), buffer, gettext ("no maximum range"));
00707           msg = buffer2;
00708           goto exit_on_error;
00709         }
00710       if (input->rangemax[input->nvariables]
00711           < input->rangemin[input->nvariables])
00712         {
00713           snprintf (buffer2, 64, "%s %s: %s",
00714                     gettext ("Variable"), buffer, gettext ("bad range"));
00715           msg = buffer2;
00716           goto exit_on_error;
00717         }
00718       input->precision = g_realloc
00719         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00720       input->precision[input->nvariables]
00721         = xml_node_get_uint_with_default (child,
    XML_PRECISION,
00722                                           DEFAULT_PRECISION, &error_code);
00723       if (error_code || input->precision[input->nvariables] >=
    NPRECISIONS)
00724         {
00725           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00726                     gettext ("bad precision"));
```

```
00727           msg = buffer2;
00728           goto exit_on_error;
00729         }
00730       if (input->algorithm == ALGORITHM_SWEEP)
00731         {
00732           if (xmlHasProp (child, XML_NSWEEPS))
00733             {
00734               input->nsweeps = (unsigned int *)
00735                 g_realloc (input->nsweeps,
00736                            (1 + input->nvariables) * sizeof (unsigned int));
00737               input->nsweeps[input->nvariables]
00738                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00739               if (error_code || !input->nsweeps[input->
      nvariables])
00740                 {
00741                   snprintf (buffer2, 64, "%s %s: %s",
00742                             gettext ("Variable"),
00743                             buffer, gettext ("bad sweeps"));
00744                   msg = buffer2;
00745                   goto exit_on_error;
00746                 }
00747             }
00748           else
00749             {
00750               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00751                         gettext ("no sweeps number"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755 #if DEBUG
00756           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00757                    input->nsweeps[input->nvariables],
      input->nsimulations);
00758 #endif
00759         }
00760       if (input->algorithm == ALGORITHM_GENETIC)
00761         {
00762           // Obtaining bits representing each variable
00763           if (xmlHasProp (child, XML_NBITS))
00764             {
00765               input->nbits = (unsigned int *)
00766                 g_realloc (input->nbits,
00767                            (1 + input->nvariables) * sizeof (unsigned int));
00768               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00769               if (error_code || !i)
00770                 {
00771                   snprintf (buffer2, 64, "%s %s: %s",
00772                             gettext ("Variable"),
00773                             buffer, gettext ("invalid bits number"));
00774                   msg = buffer2;
00775                   goto exit_on_error;
00776                 }
00777               input->nbits[input->nvariables] = i;
00778             }
00779           else
00780             {
00781               snprintf (buffer2, 64, "%s %s: %s",
00782                         gettext ("Variable"),
00783                         buffer, gettext ("no bits number"));
00784               msg = buffer2;
00785               goto exit_on_error;
00786             }
00787         }
00788       else if (input->nsteps)
00789         {
00790           input->step = (double *)
00791             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
00792           input->step[input->nvariables]
00793             = xml_node_get_float (child, XML_STEP, &error_code);
00794           if (error_code || input->step[input->nvariables] < 0.)
00795             {
00796               snprintf (buffer2, 64, "%s %s: %s",
00797                         gettext ("Variable"),
00798                         buffer, gettext ("bad step size"));
00799               msg = buffer2;
00800               goto exit_on_error;
00801             }
00802         }
00803       input->label = g_realloc
00804         (input->label, (1 + input->nvariables) * sizeof (char *));
00805       input->label[input->nvariables] = (char *) buffer;
00806       ++input->nvariables;
00807     }
00808   if (!input->nvariables)
00809     {
00810       msg = gettext ("No optimization variables");
00811       goto exit_on_error;
```

```
00812     }
00813   buffer = NULL;
00814
00815   // Obtaining the error norm
00816   if (xmlHasProp (node, XML_NORM))
00817     {
00818       buffer = xmlGetProp (node, XML_NORM);
00819       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00820         input->norm = ERROR_NORM_EUCLIDIAN;
00821       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00822         input->norm = ERROR_NORM_MAXIMUM;
00823       else if (!xmlStrcmp (buffer, XML_P))
00824         {
00825           input->norm = ERROR_NORM_P;
00826           input->p = xml_node_get_float (node, XML_P, &error_code);
00827           if (!error_code)
00828             {
00829               msg = gettext ("Bad P parameter");
00830               goto exit_on_error;
00831             }
00832         }
00833       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00834         input->norm = ERROR_NORM_TAXICAB;
00835       else
00836         {
00837           msg = gettext ("Unknown error norm");
00838           goto exit_on_error;
00839         }
00840       xmlFree (buffer);
00841     }
00842   else
00843     input->norm = ERROR_NORM_EUCLIDIAN;
00844
00845   // Getting the working directory
00846   input->directory = g_path_get_dirname (filename);
00847   input->name = g_path_get_basename (filename);
00848
00849   // Closing the XML document
00850   xmlFreeDoc (doc);
00851
00852 #if DEBUG
00853   fprintf (stderr, "input_open: end\n");
00854 #endif
00855   return 1;
00856
00857 exit_on_error:
00858   xmlFree (buffer);
00859   xmlFreeDoc (doc);
00860   show_error (msg);
00861   input_free ();
00862 #if DEBUG
00863   fprintf (stderr, "input_open: end\n");
00864 #endif
00865   return 0;
00866 }
```

Here is the call graph for this function:

5.9.2.2 void optimize_best (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1236 of file optimize.c.

```
01237 {
01238   unsigned int i, j;
01239   double e;
01240 #if DEBUG
01241   fprintf (stderr, "optimize_best: start\n");
01242   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
01243            optimize->nsaveds, optimize->nbest);
01244 #endif
01245   if (optimize->nsaveds < optimize->nbest
01246       || value < optimize->error_best[optimize->nsaveds - 1])
01247     {
```

```
01248       if (optimize->nsaveds < optimize->nbest)
01249         ++optimize->nsaveds;
01250       optimize->error_best[optimize->nsaveds - 1] = value;
01251       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
01252       for (i = optimize->nsaveds; --i;)
01253         {
01254           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
01255             {
01256               j = optimize->simulation_best[i];
01257               e = optimize->error_best[i];
01258               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
01259               optimize->error_best[i] = optimize->
      error_best[i - 1];
01260               optimize->simulation_best[i - 1] = j;
01261               optimize->error_best[i - 1] = e;
01262             }
01263           else
01264             break;
01265         }
01266     }
01267 #if DEBUG
01268   fprintf (stderr, "optimize_best: end\n");
01269 #endif
01270 }
```

5.9.2.3 void optimize_best_direction (unsigned int *simulation*, double *value*)

Function to save the best simulation in a direction search method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1561 of file optimize.c.

```
01562 {
01563 #if DEBUG
01564   fprintf (stderr, "optimize_best_direction: start\n");
01565   fprintf (stderr,
01566            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
01567            simulation, value, optimize->error_best[0]);
01568 #endif
01569   if (value < optimize->error_best[0])
01570     {
01571       optimize->error_best[0] = value;
01572       optimize->simulation_best[0] = simulation;
01573 #if DEBUG
01574       fprintf (stderr,
01575                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
01576                simulation, value);
01577 #endif
01578     }
01579 #if DEBUG
01580   fprintf (stderr, "optimize_best_direction: end\n");
01581 #endif
01582 }
```

5.9.2.4 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

simulation	Simulation number.

Definition at line 1591 of file optimize.c.

```
01592 {
01593   unsigned int i, j;
01594   double e;
01595 #if DEBUG
```

```
01596   fprintf (stderr, "optimize_direction_sequential: start\n");
01597   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
01598            "nend_direction=%u\n",
01599            optimize->nstart_direction, optimize->
      nend_direction);
01600 #endif
01601   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
01602     {
01603       j = simulation + i;
01604       e = optimize_norm (j);
01605       optimize_best_direction (j, e);
01606       optimize_save_variables (j, e);
01607       if (e < optimize->thresold)
01608         {
01609           optimize->stop = 1;
01610           break;
01611         }
01612 #if DEBUG
01613       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
01614 #endif
01615     }
01616 #if DEBUG
01617   fprintf (stderr, "optimize_direction_sequential: end\n");
01618 #endif
01619 }
```

Here is the call graph for this function:

5.9.2.5 void * optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1629 of file optimize.c.

```
01630 {
01631   unsigned int i, thread;
01632   double e;
01633 #if DEBUG
01634   fprintf (stderr, "optimize_direction_thread: start\n");
01635 #endif
01636   thread = data->thread;
01637 #if DEBUG
01638   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
01639            thread,
01640            optimize->thread_direction[thread],
01641            optimize->thread_direction[thread + 1]);
01642 #endif
01643   for (i = optimize->thread_direction[thread];
01644        i < optimize->thread_direction[thread + 1]; ++i)
01645     {
01646       e = optimize_norm (i);
01647       g_mutex_lock (mutex);
01648       optimize_best_direction (i, e);
01649       optimize_save_variables (i, e);
01650       if (e < optimize->thresold)
01651         optimize->stop = 1;
01652       g_mutex_unlock (mutex);
01653       if (optimize->stop)
01654         break;
01655 #if DEBUG
01656       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
01657 #endif
01658     }
01659 #if DEBUG
01660   fprintf (stderr, "optimize_direction_thread: end\n");
01661 #endif
01662   g_thread_exit (NULL);
01663   return NULL;
01664 }
```

Here is the call graph for this function:

5.9.2.6 double optimize_estimate_direction_coordinates (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1703 of file optimize.c.

```
01705 {
01706   double x;
01707 #if DEBUG
01708   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
01709 #endif
01710   x = optimize->direction[variable];
01711   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01712     {
01713       if (estimate & 1)
01714         x += optimize->step[variable];
01715       else
01716         x -= optimize->step[variable];
01717     }
01718 #if DEBUG
01719   fprintf (stderr,
01720            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
01721            variable, x);
01722   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
01723 #endif
01724   return x;
01725 }
```

5.9.2.7 double optimize_estimate_direction_random (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1676 of file optimize.c.

```
01678 {
01679   double x;
01680 #if DEBUG
01681   fprintf (stderr, "optimize_estimate_direction_random: start\n");
01682 #endif
01683   x = optimize->direction[variable]
01684     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
01685 #if DEBUG
01686   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
01687            variable, x);
01688   fprintf (stderr, "optimize_estimate_direction_random: end\n");
01689 #endif
01690   return x;
01691 }
```

5.9.2.8 double optimize_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

Returns

objective function value.

Definition at line 1870 of file optimize.c.

```
01871 {
01872   unsigned int j;
01873   double objective;
01874   char buffer[64];
01875 #if DEBUG
01876   fprintf (stderr, "optimize_genetic_objective: start\n");
01877 #endif
01878   for (j = 0; j < optimize->nvariables; ++j)
01879     {
01880       optimize->value[entity->id * optimize->nvariables + j]
01881         = genetic_get_variable (entity, optimize->genetic_variable + j);
01882     }
01883   objective = optimize_norm (entity->id);
01884   g_mutex_lock (mutex);
01885   for (j = 0; j < optimize->nvariables; ++j)
01886     {
01887       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01888       fprintf (optimize->file_variables, buffer,
01889                genetic_get_variable (entity, optimize->genetic_variable + j));
01890     }
01891   fprintf (optimize->file_variables, "%.14le\n", objective);
01892   g_mutex_unlock (mutex);
01893 #if DEBUG
01894   fprintf (stderr, "optimize_genetic_objective: end\n");
01895 #endif
01896   return objective;
01897 }
```

5.9.2.9 void optimize_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 880 of file optimize.c.

```
00881 {
00882   unsigned int i;
00883   char buffer[32], value[32], *buffer2, *buffer3, *content;
00884   FILE *file;
00885   gsize length;
00886   GRegex *regex;
00887
00888 #if DEBUG
00889   fprintf (stderr, "optimize_input: start\n");
00890 #endif
00891
00892   // Checking the file
00893   if (!template)
00894     goto optimize_input_end;
00895
00896   // Opening template
00897   content = g_mapped_file_get_contents (template);
00898   length = g_mapped_file_get_length (template);
00899 #if DEBUG
00900   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00901 #endif
00902   file = g_fopen (input, "w");
00903
00904   // Parsing template
00905   for (i = 0; i < optimize->nvariables; ++i)
00906     {
00907 #if DEBUG
```

```
00908       fprintf (stderr, "optimize_input: variable=%u\n", i);
00909 #endif
00910       snprintf (buffer, 32, "@variable%u@", i + 1);
00911       regex = g_regex_new (buffer, 0, 0, NULL);
00912       if (i == 0)
00913         {
00914           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00915                                              optimize->label[i], 0, NULL);
00916 #if DEBUG
00917           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00918 #endif
00919         }
00920       else
00921         {
00922           length = strlen (buffer3);
00923           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00924                                              optimize->label[i], 0, NULL);
00925           g_free (buffer3);
00926         }
00927       g_regex_unref (regex);
00928       length = strlen (buffer2);
00929       snprintf (buffer, 32, "@value%u@", i + 1);
00930       regex = g_regex_new (buffer, 0, 0, NULL);
00931       snprintf (value, 32, format[optimize->precision[i]],
00932                 optimize->value[simulation * optimize->
    nvariables + i]);
00933
00934 #if DEBUG
00935       fprintf (stderr, "optimize_input: value=%s\n", value);
00936 #endif
00937       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00938                                          0, NULL);
00939       g_free (buffer2);
00940       g_regex_unref (regex);
00941     }
00942
00943   // Saving input file
00944   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00945   g_free (buffer3);
00946   fclose (file);
00947
00948 optimize_input_end:
00949 #if DEBUG
00950   fprintf (stderr, "optimize_input: end\n");
00951 #endif
00952   return;
00953 }
```

5.9.2.10 void optimize_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 optimization results.

Parameters

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1359 of file optimize.c.

```
01361 {
01362   unsigned int i, j, k, s[optimize->nbest];
01363   double e[optimize->nbest];
01364 #if DEBUG
01365   fprintf (stderr, "optimize_merge: start\n");
01366 #endif
01367   i = j = k = 0;
01368   do
01369     {
01370       if (i == optimize->nsaveds)
01371         {
01372           s[k] = simulation_best[j];
01373           e[k] = error_best[j];
01374           ++j;
01375           ++k;
01376           if (j == nsaveds)
01377             break;
01378         }
01379       else if (j == nsaveds)
01380         {
```

```
01381           s[k] = optimize->simulation_best[i];
01382           e[k] = optimize->error_best[i];
01383           ++i;
01384           ++k;
01385           if (i == optimize->nsaveds)
01386             break;
01387         }
01388       else if (optimize->error_best[i] > error_best[j])
01389         {
01390           s[k] = simulation_best[j];
01391           e[k] = error_best[j];
01392           ++j;
01393           ++k;
01394         }
01395       else
01396         {
01397           s[k] = optimize->simulation_best[i];
01398           e[k] = optimize->error_best[i];
01399           ++i;
01400           ++k;
01401         }
01402     }
01403   while (k < optimize->nbest);
01404   optimize->nsaveds = k;
01405   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
01406   memcpy (optimize->error_best, e, k * sizeof (double));
01407 #if DEBUG
01408   fprintf (stderr, "optimize_merge: end\n");
01409 #endif
01410 }
```

5.9.2.11 double optimize_norm_euclidian (unsigned int *simulation*)

Function to calculate the Euclidian error norm.

Parameters

simulation	simulation number.

Returns

Euclidian error norm.

Definition at line 1069 of file optimize.c.

```
01070 {
01071   double e, ei;
01072   unsigned int i;
01073 #if DEBUG
01074   fprintf (stderr, "optimize_norm_euclidian: start\n");
01075 #endif
01076   e = 0.;
01077   for (i = 0; i < optimize->nexperiments; ++i)
01078     {
01079       ei = optimize_parse (simulation, i);
01080       e += ei * ei;
01081     }
01082   e = sqrt (e);
01083 #if DEBUG
01084   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
01085   fprintf (stderr, "optimize_norm_euclidian: end\n");
01086 #endif
01087   return e;
01088 }
```

Here is the call graph for this function:

5.9.2.12 double optimize_norm_maximum (unsigned int *simulation*)

Function to calculate the maximum error norm.

Parameters

simulation	simulation number.

Returns

Maximum error norm.

Definition at line 1098 of file optimize.c.

```
01099 {
01100   double e, ei;
01101   unsigned int i;
01102 #if DEBUG
01103   fprintf (stderr, "optimize_norm_maximum: start\n");
01104 #endif
01105   e = 0.;
01106   for (i = 0; i < optimize->nexperiments; ++i)
01107     {
01108       ei = fabs (optimize_parse (simulation, i));
01109       e = fmax (e, ei);
01110     }
01111 #if DEBUG
01112   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
01113   fprintf (stderr, "optimize_norm_maximum: end\n");
01114 #endif
01115   return e;
01116 }
```

Here is the call graph for this function:

5.9.2.13 double optimize_norm_p (unsigned int *simulation*)

Function to calculate the P error norm.

Parameters

simulation	simulation number.

Returns

P error norm.

Definition at line 1126 of file optimize.c.

```
01127 {
01128   double e, ei;
01129   unsigned int i;
01130 #if DEBUG
01131   fprintf (stderr, "optimize_norm_p: start\n");
01132 #endif
01133   e = 0.;
01134   for (i = 0; i < optimize->nexperiments; ++i)
01135     {
01136       ei = fabs (optimize_parse (simulation, i));
01137       e += pow (ei, optimize->p);
01138     }
01139   e = pow (e, 1. / optimize->p);
01140 #if DEBUG
01141   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
01142   fprintf (stderr, "optimize_norm_p: end\n");
01143 #endif
01144   return e;
01145 }
```

Here is the call graph for this function:

5.9.2.14 double optimize_norm_taxicab (unsigned int *simulation*)

Function to calculate the taxicab error norm.

Parameters

simulation	simulation number.

Returns

Taxicab error norm.

Definition at line 1155 of file optimize.c.

```
01156 {
01157   double e;
01158   unsigned int i;
01159 #if DEBUG
01160   fprintf (stderr, "optimize_norm_taxicab: start\n");
01161 #endif
01162   e = 0.;
01163   for (i = 0; i < optimize->nexperiments; ++i)
01164     e += fabs (optimize_parse (simulation, i));
01165 #if DEBUG
01166   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
01167   fprintf (stderr, "optimize_norm_taxicab: end\n");
01168 #endif
01169   return e;
01170 }
```

Here is the call graph for this function:

5.9.2.15 double optimize_parse (unsigned int *simulation*, unsigned int *experiment*)

Function to parse input files, simulating and calculating the \ objective function.

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 966 of file optimize.c.

```
00967 {
00968   unsigned int i;
00969   double e;
00970   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00971     *buffer3, *buffer4;
00972   FILE *file_result;
00973
00974 #if DEBUG
00975   fprintf (stderr, "optimize_parse: start\n");
00976   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00977            experiment);
00978 #endif
00979
00980   // Opening input files
00981   for (i = 0; i < optimize->ninputs; ++i)
00982     {
00983       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00984 #if DEBUG
00985       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00986 #endif
00987       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00988     }
00989   for (; i < MAX_NINPUTS; ++i)
00990     strcpy (&input[i][0], "");
00991 #if DEBUG
00992   fprintf (stderr, "optimize_parse: parsing end\n");
00993 #endif
00994
00995   // Performing the simulation
```

```
00996   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00997   buffer2 = g_path_get_dirname (optimize->simulator);
00998   buffer3 = g_path_get_basename (optimize->simulator);
00999   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01000   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01001             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01002             input[6], input[7], output);
01003   g_free (buffer4);
01004   g_free (buffer3);
01005   g_free (buffer2);
01006 #if DEBUG
01007   fprintf (stderr, "optimize_parse: %s\n", buffer);
01008 #endif
01009   system (buffer);
01010
01011   // Checking the objective value function
01012   if (optimize->evaluator)
01013     {
01014       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01015       buffer2 = g_path_get_dirname (optimize->evaluator);
01016       buffer3 = g_path_get_basename (optimize->evaluator);
01017       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01018       snprintf (buffer, 512, "\"%s\" %s %s %s",
01019                 buffer4, output, optimize->experiment[experiment], result);
01020       g_free (buffer4);
01021       g_free (buffer3);
01022       g_free (buffer2);
01023 #if DEBUG
01024       fprintf (stderr, "optimize_parse: %s\n", buffer);
01025 #endif
01026       system (buffer);
01027       file_result = g_fopen (result, "r");
01028       e = atof (fgets (buffer, 512, file_result));
01029       fclose (file_result);
01030     }
01031   else
01032     {
01033       strcpy (result, "");
01034       file_result = g_fopen (output, "r");
01035       e = atof (fgets (buffer, 512, file_result));
01036       fclose (file_result);
01037     }
01038
01039   // Removing files
01040 #if !DEBUG
01041   for (i = 0; i < optimize->ninputs; ++i)
01042     {
01043       if (optimize->file[i][0])
01044         {
01045           snprintf (buffer, 512, RM " %s", &input[i][0]);
01046           system (buffer);
01047         }
01048     }
01049   snprintf (buffer, 512, RM " %s %s", output, result);
01050   system (buffer);
01051 #endif
01052
01053 #if DEBUG
01054   fprintf (stderr, "optimize_parse: end\n");
01055 #endif
01056
01057   // Returning the objective function
01058   return e * optimize->weight[experiment];
01059 }
```

Here is the call graph for this function:

5.9.2.16 void optimize_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1208 of file optimize.c.

```
01209 {
01210   unsigned int i;
```

```
01211   char buffer[64];
01212 #if DEBUG
01213   fprintf (stderr, "optimize_save_variables: start\n");
01214 #endif
01215   for (i = 0; i < optimize->nvariables; ++i)
01216     {
01217       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
01218       fprintf (optimize->file_variables, buffer,
01219                optimize->value[simulation * optimize->
      nvariables + i]);
01220     }
01221   fprintf (optimize->file_variables, "%.14le\n", error);
01222 #if DEBUG
01223   fprintf (stderr, "optimize_save_variables: end\n");
01224 #endif
01225 }
```

5.9.2.17 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 1734 of file optimize.c.

```
01735 {
01736   GThread *thread[nthreads_direction];
01737   ParallelData data[nthreads_direction];
01738   unsigned int i, j, k, b;
01739 #if DEBUG
01740   fprintf (stderr, "optimize_step_direction: start\n");
01741 #endif
01742   for (i = 0; i < optimize->nestimates; ++i)
01743     {
01744       k = (simulation + i) * optimize->nvariables;
01745       b = optimize->simulation_best[0] * optimize->
      nvariables;
01746 #if DEBUG
01747       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
01748                simulation + i, optimize->simulation_best[0]);
01749 #endif
01750       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
01751         {
01752 #if DEBUG
01753           fprintf (stderr,
01754                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
01755                    i, j, optimize->value[b]);
01756 #endif
01757           optimize->value[k]
01758             = optimize->value[b] + optimize_estimate_direction (j,
      i);
01759           optimize->value[k] = fmin (fmax (optimize->value[k],
01760                                            optimize->rangeminabs[j]),
01761                                      optimize->rangemaxabs[j]);
01762 #if DEBUG
01763           fprintf (stderr,
01764                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
01765                    i, j, optimize->value[k]);
01766 #endif
01767         }
01768     }
01769   if (nthreads_direction == 1)
01770     optimize_direction_sequential (simulation);
01771   else
01772     {
01773       for (i = 0; i <= nthreads_direction; ++i)
01774         {
01775           optimize->thread_direction[i]
01776             = simulation + optimize->nstart_direction
01777             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01778               / nthreads_direction;
01779 #if DEBUG
01780           fprintf (stderr,
01781                    "optimize_step_direction: i=%u thread_direction=%u\n",
01782                    i, optimize->thread_direction[i]);
01783 #endif
01784         }
01785       for (i = 0; i < nthreads_direction; ++i)
```

```
01786         {
01787           data[i].thread = i;
01788           thread[i] = g_thread_new
01789             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01790         }
01791       for (i = 0; i < nthreads_direction; ++i)
01792         g_thread_join (thread[i]);
01793     }
01794 #if DEBUG
01795   fprintf (stderr, "optimize_step_direction: end\n");
01796 #endif
01797 }
```

Here is the call graph for this function:

5.9.2.18 void * optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1313 of file optimize.c.

```
01314 {
01315   unsigned int i, thread;
01316   double e;
01317 #if DEBUG
01318   fprintf (stderr, "optimize_thread: start\n");
01319 #endif
01320   thread = data->thread;
01321 #if DEBUG
01322   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
01323            optimize->thread[thread], optimize->thread[thread + 1]);
01324 #endif
01325   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
01326     {
01327       e = optimize_norm (i);
01328       g_mutex_lock (mutex);
01329       optimize_best (i, e);
01330       optimize_save_variables (i, e);
01331       if (e < optimize->thresold)
01332         optimize->stop = 1;
01333       g_mutex_unlock (mutex);
01334       if (optimize->stop)
01335         break;
01336 #if DEBUG
01337       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
01338 #endif
01339     }
01340 #if DEBUG
01341   fprintf (stderr, "optimize_thread: end\n");
01342 #endif
01343   g_thread_exit (NULL);
01344   return NULL;
01345 }
```

Here is the call graph for this function:

5.9.3 Variable Documentation

5.9.3.1 const char* format[NPRECISIONS]

Initial value:

```
= {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
}
```

Array of C-strings with variable formats.

Definition at line 101 of file optimize.c.

5.9.3.2 const double precision[NPRECISIONS]

Initial value:

```
= {
  1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
  1e-13, 1e-14
}
```

Array of variable precisions.

Definition at line 106 of file optimize.c.

5.9.3.3 const xmlChar* template[MAX_NINPUTS]

Initial value:

```
= {
  XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
  XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
}
```

Array of xmlChar strings with template labels.

Definition at line 94 of file optimize.c.

5.10 optimize.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
```

```
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elif !defined (BSD)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "utils.h"
00059 #include "optimize.h"
00060
00061 #define DEBUG 0
00062
00063
00067 #ifdef G_OS_WIN32
00068 #define RM "del"
00069 #else
00070 #define RM "rm"
00071 #endif
00072
00073 int ntasks;
00074 unsigned int nthreads;
00075 unsigned int nthreads_direction;
00077 GMutex mutex[1];
00078 void (*optimize_algorithm) ();
00080 double (*optimize_estimate_direction) (unsigned int variable,
00081                                        unsigned int estimate);
00083 double (*optimize_norm) (unsigned int simulation);
00085 Input input[1];
00087 Optimize optimize[1];
00088
00089 const xmlChar *result_name = (xmlChar *) "result";
00091 const xmlChar *variables_name = (xmlChar *) "variables";
00093
00094 const xmlChar *template[MAX_NINPUTS] = {
00095   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00096   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00097 };
00098
00100
00101 const char *format[NPRECISIONS] = {
00102   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00103   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00104 };
00105
00106 const double precision[NPRECISIONS] = {
00107   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00108   1e-13, 1e-14
00109 };
00110
00115 void
00116 input_new ()
00117 {
00118   unsigned int i;
00119 #if DEBUG
00120   fprintf (stderr, "input_new: start\n");
00121 #endif
00122   input->nvariables = input->nexperiments = input->ninputs = input->
      nsteps = 0;
00123   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00124   input->experiment = input->label = NULL;
00125   input->precision = input->nsweeps = input->nbits = NULL;
00126   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00127     = input->weight = input->step = NULL;
00128   for (i = 0; i < MAX_NINPUTS; ++i)
00129     input->template[i] = NULL;
00130 #if DEBUG
00131   fprintf (stderr, "input_new: end\n");
00132 #endif
00133 }
00134
00139 void
00140 input_free ()
00141 {
00142   unsigned int i, j;
00143 #if DEBUG
00144   fprintf (stderr, "input_free: start\n");
00145 #endif
```

```
00146   g_free (input->name);
00147   g_free (input->directory);
00148   for (i = 0; i < input->nexperiments; ++i)
00149     {
00150       xmlFree (input->experiment[i]);
00151       for (j = 0; j < input->ninputs; ++j)
00152         xmlFree (input->template[j][i]);
00153       g_free (input->template[j]);
00154     }
00155   g_free (input->experiment);
00156   for (i = 0; i < input->ninputs; ++i)
00157     g_free (input->template[i]);
00158   for (i = 0; i < input->nvariables; ++i)
00159     xmlFree (input->label[i]);
00160   g_free (input->label);
00161   g_free (input->precision);
00162   g_free (input->rangemin);
00163   g_free (input->rangemax);
00164   g_free (input->rangeminabs);
00165   g_free (input->rangemaxabs);
00166   g_free (input->weight);
00167   g_free (input->step);
00168   g_free (input->nsweeps);
00169   g_free (input->nbits);
00170   xmlFree (input->evaluator);
00171   xmlFree (input->simulator);
00172   xmlFree (input->result);
00173   xmlFree (input->variables);
00174   input->nexperiments = input->ninputs = input->nvariables = input->
      nsteps = 0;
00175 #if DEBUG
00176   fprintf (stderr, "input_free: end\n");
00177 #endif
00178 }
00179
00187 int
00188 input_open (char *filename)
00189 {
00190   char buffer2[64];
00191   char *buffert[MAX_NINPUTS] =
00192     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00193   xmlDoc *doc;
00194   xmlNode *node, *child;
00195   xmlChar *buffer;
00196   char *msg;
00197   int error_code;
00198   unsigned int i;
00199
00200 #if DEBUG
00201   fprintf (stderr, "input_open: start\n");
00202 #endif
00203
00204   // Resetting input data
00205   buffer = NULL;
00206   input_new ();
00207
00208   // Parsing the input file
00209 #if DEBUG
00210   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00211 #endif
00212   doc = xmlParseFile (filename);
00213   if (!doc)
00214     {
00215       msg = gettext ("Unable to parse the input file");
00216       goto exit_on_error;
00217     }
00218
00219   // Getting the root node
00220 #if DEBUG
00221   fprintf (stderr, "input_open: getting the root node\n");
00222 #endif
00223   node = xmlDocGetRootElement (doc);
00224   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00225     {
00226       msg = gettext ("Bad root XML node");
00227       goto exit_on_error;
00228     }
00229
00230   // Getting result and variables file names
00231   if (!input->result)
00232     {
00233       input->result = (char *) xmlGetProp (node, XML_RESULT);
00234       if (!input->result)
00235         input->result = (char *) xmlStrdup (result_name);
00236     }
00237   if (!input->variables)
00238     {
```

```
00239       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00240       if (!input->variables)
00241         input->variables = (char *) xmlStrdup (variables_name);
00242     }
00243
00244   // Opening simulator program name
00245   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00246   if (!input->simulator)
00247     {
00248       msg = gettext ("Bad simulator program");
00249       goto exit_on_error;
00250     }
00251
00252   // Opening evaluator program name
00253   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00254
00255   // Obtaining pseudo-random numbers generator seed
00256   input->seed
00257     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00258                                       &error_code);
00259   if (error_code)
00260     {
00261       msg = gettext ("Bad pseudo-random numbers generator seed");
00262       goto exit_on_error;
00263     }
00264
00265   // Opening algorithm
00266   buffer = xmlGetProp (node, XML_ALGORITHM);
00267   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00268     {
00269       input->algorithm = ALGORITHM_MONTE_CARLO;
00270
00271       // Obtaining simulations number
00272       input->nsimulations
00273         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00274       if (error_code)
00275         {
00276           msg = gettext ("Bad simulations number");
00277           goto exit_on_error;
00278         }
00279     }
00280   else if (!xmlStrcmp (buffer, XML_SWEEP))
00281     input->algorithm = ALGORITHM_SWEEP;
00282   else if (!xmlStrcmp (buffer, XML_GENETIC))
00283     {
00284       input->algorithm = ALGORITHM_GENETIC;
00285
00286       // Obtaining population
00287       if (xmlHasProp (node, XML_NPOPULATION))
00288         {
00289           input->nsimulations
00290             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00291           if (error_code || input->nsimulations < 3)
00292             {
00293               msg = gettext ("Invalid population number");
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           msg = gettext ("No population number");
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining generations
00304       if (xmlHasProp (node, XML_NGENERATIONS))
00305         {
00306           input->niterations
00307             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00308           if (error_code || !input->niterations)
00309             {
00310               msg = gettext ("Invalid generations number");
00311               goto exit_on_error;
00312             }
00313         }
00314       else
00315         {
00316           msg = gettext ("No generations number");
00317           goto exit_on_error;
00318         }
00319
00320       // Obtaining mutation probability
00321       if (xmlHasProp (node, XML_MUTATION))
00322         {
00323           input->mutation_ratio
00324             = xml_node_get_float (node, XML_MUTATION, &error_code);
```

```
00325           if (error_code || input->mutation_ratio < 0.
00326               || input->mutation_ratio >= 1.)
00327             {
00328               msg = gettext ("Invalid mutation probability");
00329               goto exit_on_error;
00330             }
00331         }
00332       else
00333         {
00334           msg = gettext ("No mutation probability");
00335           goto exit_on_error;
00336         }
00337
00338       // Obtaining reproduction probability
00339       if (xmlHasProp (node, XML_REPRODUCTION))
00340         {
00341           input->reproduction_ratio
00342             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00343           if (error_code || input->reproduction_ratio < 0.
00344               || input->reproduction_ratio >= 1.0)
00345             {
00346               msg = gettext ("Invalid reproduction probability");
00347               goto exit_on_error;
00348             }
00349         }
00350       else
00351         {
00352           msg = gettext ("No reproduction probability");
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining adaptation probability
00357       if (xmlHasProp (node, XML_ADAPTATION))
00358         {
00359           input->adaptation_ratio
00360             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00361           if (error_code || input->adaptation_ratio < 0.
00362               || input->adaptation_ratio >= 1.)
00363             {
00364               msg = gettext ("Invalid adaptation probability");
00365               goto exit_on_error;
00366             }
00367         }
00368       else
00369         {
00370           msg = gettext ("No adaptation probability");
00371           goto exit_on_error;
00372         }
00373
00374       // Checking survivals
00375       i = input->mutation_ratio * input->nsimulations;
00376       i += input->reproduction_ratio * input->nsimulations;
00377       i += input->adaptation_ratio * input->nsimulations;
00378       if (i > input->nsimulations - 2)
00379         {
00380           msg = gettext
00381             ("No enough survival entities to reproduce the population");
00382           goto exit_on_error;
00383         }
00384     }
00385   else
00386     {
00387       msg = gettext ("Unknown algorithm");
00388       goto exit_on_error;
00389     }
00390   xmlFree (buffer);
00391   buffer = NULL;
00392
00393   if (input->algorithm == ALGORITHM_MONTE_CARLO
00394       || input->algorithm == ALGORITHM_SWEEP)
00395     {
00396
00397       // Obtaining iterations number
00398       input->niterations
00399         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00400       if (error_code == 1)
00401         input->niterations = 1;
00402       else if (error_code)
00403         {
00404           msg = gettext ("Bad iterations number");
00405           goto exit_on_error;
00406         }
00407
00408       // Obtaining best number
00409       input->nbest
00410         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
```

```
00411       if (error_code || !input->nbest)
00412         {
00413           msg = gettext ("Invalid best number");
00414           goto exit_on_error;
00415         }
00416
00417       // Obtaining tolerance
00418       input->tolerance
00419         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00420                                            &error_code);
00421       if (error_code || input->tolerance < 0.)
00422         {
00423           msg = gettext ("Invalid tolerance");
00424           goto exit_on_error;
00425         }
00426
00427       // Getting direction search method parameters
00428       if (xmlHasProp (node, XML_NSTEPS))
00429         {
00430           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00431           if (error_code || !input->nsteps)
00432             {
00433               msg = gettext ("Invalid steps number");
00434               goto exit_on_error;
00435             }
00436           buffer = xmlGetProp (node, XML_DIRECTION);
00437           if (!xmlStrcmp (buffer, XML_COORDINATES))
00438             input->direction = DIRECTION_METHOD_COORDINATES;
00439           else if (!xmlStrcmp (buffer, XML_RANDOM))
00440             {
00441               input->direction = DIRECTION_METHOD_RANDOM;
00442               input->nestimates
00443                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00444               if (error_code || !input->nestimates)
00445                 {
00446                   msg = gettext ("Invalid estimates number");
00447                   goto exit_on_error;
00448                 }
00449             }
00450           else
00451             {
00452               msg = gettext ("Unknown method to estimate the direction search");
00453               goto exit_on_error;
00454             }
00455           xmlFree (buffer);
00456           buffer = NULL;
00457           input->relaxation
00458             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00459                                                DEFAULT_RELAXATION, &error_code);
00460           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00461             {
00462               msg = gettext ("Invalid relaxation parameter");
00463               goto exit_on_error;
00464             }
00465         }
00466       else
00467         input->nsteps = 0;
00468     }
00469   // Obtaining the thresold
00470   input->thresold = xml_node_get_float_with_default (node,
      XML_THRESOLD, 0.,
00471                                                      &error_code);
00472   if (error_code)
00473     {
00474       msg = gettext ("Invalid thresold");
00475       goto exit_on_error;
00476     }
00477
00478   // Reading the experimental data
00479   for (child = node->children; child; child = child->next)
00480     {
00481       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00482         break;
00483 #if DEBUG
00484       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00485 #endif
00486       if (xmlHasProp (child, XML_NAME))
00487         buffer = xmlGetProp (child, XML_NAME);
00488       else
00489         {
00490           snprintf (buffer2, 64, "%s %u: %s",
00491                     gettext ("Experiment"),
00492                     input->nexperiments + 1, gettext ("no data file name"));
```

```
00493           msg = buffer2;
00494           goto exit_on_error;
00495         }
00496 #if DEBUG
00497       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00498 #endif
00499       input->weight = g_realloc (input->weight,
00500                                  (1 + input->nexperiments) * sizeof (double));
00501       input->weight[input->nexperiments]
00502         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00503       if (error_code)
00504         {
00505           snprintf (buffer2, 64, "%s %s: %s",
00506                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00507           msg = buffer2;
00508           goto exit_on_error;
00509         }
00510 #if DEBUG
00511       fprintf (stderr, "input_open: weight=%lg\n",
00512                input->weight[input->nexperiments]);
00513 #endif
00514       if (!input->nexperiments)
00515         input->ninputs = 0;
00516 #if DEBUG
00517       fprintf (stderr, "input_open: template[0]\n");
00518 #endif
00519       if (xmlHasProp (child, XML_TEMPLATE1))
00520         {
00521           input->template[0]
00522             = (char **) g_realloc (input->template[0],
00523                                    (1 + input->nexperiments) * sizeof (char *));
00524           buffert[0] = (char *) xmlGetProp (child, template[0]);
00525 #if DEBUG
00526           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00527                    input->nexperiments, buffert[0]);
00528 #endif
00529           if (!input->nexperiments)
00530             ++input->ninputs;
00531 #if DEBUG
00532           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00533 #endif
00534         }
00535       else
00536         {
00537           snprintf (buffer2, 64, "%s %s: %s",
00538                     gettext ("Experiment"), buffer, gettext ("no template"));
00539           msg = buffer2;
00540           goto exit_on_error;
00541         }
00542       for (i = 1; i < MAX_NINPUTS; ++i)
00543         {
00544 #if DEBUG
00545           fprintf (stderr, "input_open: template%u\n", i + 1);
00546 #endif
00547           if (xmlHasProp (child, template[i]))
00548             {
00549               if (input->nexperiments && input->ninputs <= i)
00550                 {
00551                   snprintf (buffer2, 64, "%s %s: %s",
00552                             gettext ("Experiment"),
00553                             buffer, gettext ("bad templates number"));
00554                   msg = buffer2;
00555                   while (i-- > 0)
00556                     xmlFree (buffert[i]);
00557                   goto exit_on_error;
00558                 }
00559               input->template[i] = (char **)
00560                 g_realloc (input->template[i],
00561                            (1 + input->nexperiments) * sizeof (char *));
00562               buffert[i] = (char *) xmlGetProp (child, template[i]);
00563 #if DEBUG
00564               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00565                        input->nexperiments, i + 1,
00566                        input->template[i][input->nexperiments]);
00567 #endif
00568               if (!input->nexperiments)
00569                 ++input->ninputs;
00570 #if DEBUG
00571               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00572 #endif
00573             }
00574           else if (input->nexperiments && input->ninputs > i)
00575             {
00576               snprintf (buffer2, 64, "%s %s: %s%u",
00577                         gettext ("Experiment"),
00578                         buffer, gettext ("no template"), i + 1);
```

```
00579               msg = buffer2;
00580               while (i-- > 0)
00581                 xmlFree (buffert[i]);
00582               goto exit_on_error;
00583             }
00584           else
00585             break;
00586         }
00587       input->experiment
00588         = g_realloc (input->experiment,
00589                      (1 + input->nexperiments) * sizeof (char *));
00590       input->experiment[input->nexperiments] = (char *) buffer;
00591       for (i = 0; i < input->ninputs; ++i)
00592         input->template[i][input->nexperiments] = buffert[i];
00593       ++input->nexperiments;
00594 #if DEBUG
00595       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00596 #endif
00597     }
00598   if (!input->nexperiments)
00599     {
00600       msg = gettext ("No optimization experiments");
00601       goto exit_on_error;
00602     }
00603   buffer = NULL;
00604
00605   // Reading the variables data
00606   for (; child; child = child->next)
00607     {
00608       if (xmlStrcmp (child->name, XML_VARIABLE))
00609         {
00610           snprintf (buffer2, 64, "%s %u: %s",
00611                     gettext ("Variable"),
00612                     input->nvariables + 1, gettext ("bad XML node"));
00613           msg = buffer2;
00614           goto exit_on_error;
00615         }
00616       if (xmlHasProp (child, XML_NAME))
00617         buffer = xmlGetProp (child, XML_NAME);
00618       else
00619         {
00620           snprintf (buffer2, 64, "%s %u: %s",
00621                     gettext ("Variable"),
00622                     input->nvariables + 1, gettext ("no name"));
00623           msg = buffer2;
00624           goto exit_on_error;
00625         }
00626       if (xmlHasProp (child, XML_MINIMUM))
00627         {
00628           input->rangemin = g_realloc
00629             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00630           input->rangeminabs = g_realloc
00631             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00632           input->rangemin[input->nvariables]
00633             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00634           if (error_code)
00635             {
00636               snprintf (buffer2, 64, "%s %s: %s",
00637                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00638               msg = buffer2;
00639               goto exit_on_error;
00640             }
00641           input->rangeminabs[input->nvariables]
00642             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MINIMUM,
00643                                                -G_MAXDOUBLE, &error_code);
00644           if (error_code)
00645             {
00646               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00647                         gettext ("bad absolute minimum"));
00648               msg = buffer2;
00649               goto exit_on_error;
00650             }
00651           if (input->rangemin[input->nvariables]
00652               < input->rangeminabs[input->nvariables])
00653             {
00654               snprintf (buffer2, 64, "%s %s: %s",
00655                         gettext ("Variable"),
00656                         buffer, gettext ("minimum range not allowed"));
00657               msg = buffer2;
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           snprintf (buffer2, 64, "%s %s: %s",
00664                     gettext ("Variable"), buffer, gettext ("no minimum range"));
```

```
00665           msg = buffer2;
00666           goto exit_on_error;
00667         }
00668       if (xmlHasProp (child, XML_MAXIMUM))
00669         {
00670           input->rangemax = g_realloc
00671             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00672           input->rangemaxabs = g_realloc
00673             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00674           input->rangemax[input->nvariables]
00675             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00676           if (error_code)
00677             {
00678               snprintf (buffer2, 64, "%s %s: %s",
00679                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00680               msg = buffer2;
00681               goto exit_on_error;
00682             }
00683           input->rangemaxabs[input->nvariables]
00684             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MAXIMUM,
00685                                                G_MAXDOUBLE, &error_code);
00686           if (error_code)
00687             {
00688               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00689                         gettext ("bad absolute maximum"));
00690               msg = buffer2;
00691               goto exit_on_error;
00692             }
00693           if (input->rangemax[input->nvariables]
00694               > input->rangemaxabs[input->nvariables])
00695             {
00696               snprintf (buffer2, 64, "%s %s: %s",
00697                         gettext ("Variable"),
00698                         buffer, gettext ("maximum range not allowed"));
00699               msg = buffer2;
00700               goto exit_on_error;
00701             }
00702         }
00703       else
00704         {
00705           snprintf (buffer2, 64, "%s %s: %s",
00706                     gettext ("Variable"), buffer, gettext ("no maximum range"));
00707           msg = buffer2;
00708           goto exit_on_error;
00709         }
00710       if (input->rangemax[input->nvariables]
00711           < input->rangemin[input->nvariables])
00712         {
00713           snprintf (buffer2, 64, "%s %s: %s",
00714                     gettext ("Variable"), buffer, gettext ("bad range"));
00715           msg = buffer2;
00716           goto exit_on_error;
00717         }
00718       input->precision = g_realloc
00719         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00720       input->precision[input->nvariables]
00721         = xml_node_get_uint_with_default (child,
    XML_PRECISION,
00722                                           DEFAULT_PRECISION, &error_code);
00723       if (error_code || input->precision[input->nvariables] >=
    NPRECISIONS)
00724         {
00725           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00726                     gettext ("bad precision"));
00727           msg = buffer2;
00728           goto exit_on_error;
00729         }
00730       if (input->algorithm == ALGORITHM_SWEEP)
00731         {
00732           if (xmlHasProp (child, XML_NSWEEPS))
00733             {
00734               input->nsweeps = (unsigned int *)
00735                 g_realloc (input->nsweeps,
00736                            (1 + input->nvariables) * sizeof (unsigned int));
00737               input->nsweeps[input->nvariables]
00738                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00739               if (error_code || !input->nsweeps[input->nvariables])
00740                 {
00741                   snprintf (buffer2, 64, "%s %s: %s",
00742                             gettext ("Variable"),
00743                             buffer, gettext ("bad sweeps"));
00744                   msg = buffer2;
00745                   goto exit_on_error;
00746                 }
00747             }
00748           else
```

```
00749             {
00750               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00751                         gettext ("no sweeps number"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755 #if DEBUG
00756           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00757                    input->nsweeps[input->nvariables], input->
      nsimulations);
00758 #endif
00759         }
00760       if (input->algorithm == ALGORITHM_GENETIC)
00761         {
00762           // Obtaining bits representing each variable
00763           if (xmlHasProp (child, XML_NBITS))
00764             {
00765               input->nbits = (unsigned int *)
00766                 g_realloc (input->nbits,
00767                            (1 + input->nvariables) * sizeof (unsigned int));
00768               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00769               if (error_code || !i)
00770                 {
00771                   snprintf (buffer2, 64, "%s %s: %s",
00772                             gettext ("Variable"),
00773                             buffer, gettext ("invalid bits number"));
00774                   msg = buffer2;
00775                   goto exit_on_error;
00776                 }
00777               input->nbits[input->nvariables] = i;
00778             }
00779           else
00780             {
00781               snprintf (buffer2, 64, "%s %s: %s",
00782                         gettext ("Variable"),
00783                         buffer, gettext ("no bits number"));
00784               msg = buffer2;
00785               goto exit_on_error;
00786             }
00787         }
00788       else if (input->nsteps)
00789         {
00790           input->step = (double *)
00791             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
00792           input->step[input->nvariables]
00793             = xml_node_get_float (child, XML_STEP, &error_code);
00794           if (error_code || input->step[input->nvariables] < 0.)
00795             {
00796               snprintf (buffer2, 64, "%s %s: %s",
00797                         gettext ("Variable"),
00798                         buffer, gettext ("bad step size"));
00799               msg = buffer2;
00800               goto exit_on_error;
00801             }
00802         }
00803       input->label = g_realloc
00804         (input->label, (1 + input->nvariables) * sizeof (char *));
00805       input->label[input->nvariables] = (char *) buffer;
00806       ++input->nvariables;
00807     }
00808   if (!input->nvariables)
00809     {
00810       msg = gettext ("No optimization variables");
00811       goto exit_on_error;
00812     }
00813   buffer = NULL;
00814
00815   // Obtaining the error norm
00816   if (xmlHasProp (node, XML_NORM))
00817     {
00818       buffer = xmlGetProp (node, XML_NORM);
00819       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00820         input->norm = ERROR_NORM_EUCLIDIAN;
00821       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00822         input->norm = ERROR_NORM_MAXIMUM;
00823       else if (!xmlStrcmp (buffer, XML_P))
00824         {
00825           input->norm = ERROR_NORM_P;
00826           input->p = xml_node_get_float (node, XML_P, &error_code);
00827           if (!error_code)
00828             {
00829               msg = gettext ("Bad P parameter");
00830               goto exit_on_error;
00831             }
00832         }
00833       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00834         input->norm = ERROR_NORM_TAXICAB;
```

```
00835       else
00836         {
00837           msg = gettext ("Unknown error norm");
00838           goto exit_on_error;
00839         }
00840       xmlFree (buffer);
00841     }
00842   else
00843     input->norm = ERROR_NORM_EUCLIDIAN;
00844
00845   // Getting the working directory
00846   input->directory = g_path_get_dirname (filename);
00847   input->name = g_path_get_basename (filename);
00848
00849   // Closing the XML document
00850   xmlFreeDoc (doc);
00851
00852 #if DEBUG
00853   fprintf (stderr, "input_open: end\n");
00854 #endif
00855   return 1;
00856
00857 exit_on_error:
00858   xmlFree (buffer);
00859   xmlFreeDoc (doc);
00860   show_error (msg);
00861   input_free ();
00862 #if DEBUG
00863   fprintf (stderr, "input_open: end\n");
00864 #endif
00865   return 0;
00866 }
00867
00879 void
00880 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00881 {
00882   unsigned int i;
00883   char buffer[32], value[32], *buffer2, *buffer3, *content;
00884   FILE *file;
00885   gsize length;
00886   GRegex *regex;
00887
00888 #if DEBUG
00889   fprintf (stderr, "optimize_input: start\n");
00890 #endif
00891
00892   // Checking the file
00893   if (!template)
00894     goto optimize_input_end;
00895
00896   // Opening template
00897   content = g_mapped_file_get_contents (template);
00898   length = g_mapped_file_get_length (template);
00899 #if DEBUG
00900   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00901 #endif
00902   file = g_fopen (input, "w");
00903
00904   // Parsing template
00905   for (i = 0; i < optimize->nvariables; ++i)
00906     {
00907 #if DEBUG
00908       fprintf (stderr, "optimize_input: variable=%u\n", i);
00909 #endif
00910       snprintf (buffer, 32, "@variable%u@", i + 1);
00911       regex = g_regex_new (buffer, 0, 0, NULL);
00912       if (i == 0)
00913         {
00914           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00915                                              optimize->label[i], 0, NULL);
00916 #if DEBUG
00917           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00918 #endif
00919         }
00920       else
00921         {
00922           length = strlen (buffer3);
00923           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00924                                              optimize->label[i], 0, NULL);
00925           g_free (buffer3);
00926         }
00927       g_regex_unref (regex);
00928       length = strlen (buffer2);
00929       snprintf (buffer, 32, "@value%u@", i + 1);
00930       regex = g_regex_new (buffer, 0, 0, NULL);
00931       snprintf (value, 32, format[optimize->precision[i]],
00932                 optimize->value[simulation * optimize->nvariables + i]);
```

```
00933
00934 #if DEBUG
00935       fprintf (stderr, "optimize_input: value=%s\n", value);
00936 #endif
00937       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00938                                          0, NULL);
00939       g_free (buffer2);
00940       g_regex_unref (regex);
00941     }
00942
00943   // Saving input file
00944   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00945   g_free (buffer3);
00946   fclose (file);
00947
00948 optimize_input_end:
00949 #if DEBUG
00950   fprintf (stderr, "optimize_input: end\n");
00951 #endif
00952   return;
00953 }
00954
00965 double
00966 optimize_parse (unsigned int simulation, unsigned int experiment)
00967 {
00968   unsigned int i;
00969   double e;
00970   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00971     *buffer3, *buffer4;
00972   FILE *file_result;
00973
00974 #if DEBUG
00975   fprintf (stderr, "optimize_parse: start\n");
00976   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00977            experiment);
00978 #endif
00979
00980   // Opening input files
00981   for (i = 0; i < optimize->ninputs; ++i)
00982     {
00983       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00984 #if DEBUG
00985       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00986 #endif
00987       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00988     }
00989   for (; i < MAX_NINPUTS; ++i)
00990     strcpy (&input[i][0], "");
00991 #if DEBUG
00992   fprintf (stderr, "optimize_parse: parsing end\n");
00993 #endif
00994
00995   // Performing the simulation
00996   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00997   buffer2 = g_path_get_dirname (optimize->simulator);
00998   buffer3 = g_path_get_basename (optimize->simulator);
00999   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01000   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01001             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01002             input[6], input[7], output);
01003   g_free (buffer4);
01004   g_free (buffer3);
01005   g_free (buffer2);
01006 #if DEBUG
01007   fprintf (stderr, "optimize_parse: %s\n", buffer);
01008 #endif
01009   system (buffer);
01010
01011   // Checking the objective value function
01012   if (optimize->evaluator)
01013     {
01014       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01015       buffer2 = g_path_get_dirname (optimize->evaluator);
01016       buffer3 = g_path_get_basename (optimize->evaluator);
01017       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01018       snprintf (buffer, 512, "\"%s\" %s %s %s",
01019                 buffer4, output, optimize->experiment[experiment], result);
01020       g_free (buffer4);
01021       g_free (buffer3);
01022       g_free (buffer2);
01023 #if DEBUG
01024       fprintf (stderr, "optimize_parse: %s\n", buffer);
01025 #endif
01026       system (buffer);
01027       file_result = g_fopen (result, "r");
01028       e = atof (fgets (buffer, 512, file_result));
01029       fclose (file_result);
```

```
01030     }
01031   else
01032     {
01033       strcpy (result, "");
01034       file_result = g_fopen (output, "r");
01035       e = atof (fgets (buffer, 512, file_result));
01036       fclose (file_result);
01037     }
01038
01039   // Removing files
01040 #if !DEBUG
01041   for (i = 0; i < optimize->ninputs; ++i)
01042     {
01043       if (optimize->file[i][0])
01044         {
01045           snprintf (buffer, 512, RM " %s", &input[i][0]);
01046           system (buffer);
01047         }
01048     }
01049   snprintf (buffer, 512, RM " %s %s", output, result);
01050   system (buffer);
01051 #endif
01052
01053 #if DEBUG
01054   fprintf (stderr, "optimize_parse: end\n");
01055 #endif
01056
01057   // Returning the objective function
01058   return e * optimize->weight[experiment];
01059 }
01060
01068 double
01069 optimize_norm_euclidian (unsigned int simulation)
01070 {
01071   double e, ei;
01072   unsigned int i;
01073 #if DEBUG
01074   fprintf (stderr, "optimize_norm_euclidian: start\n");
01075 #endif
01076   e = 0.;
01077   for (i = 0; i < optimize->nexperiments; ++i)
01078     {
01079       ei = optimize_parse (simulation, i);
01080       e += ei * ei;
01081     }
01082   e = sqrt (e);
01083 #if DEBUG
01084   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
01085   fprintf (stderr, "optimize_norm_euclidian: end\n");
01086 #endif
01087   return e;
01088 }
01089
01097 double
01098 optimize_norm_maximum (unsigned int simulation)
01099 {
01100   double e, ei;
01101   unsigned int i;
01102 #if DEBUG
01103   fprintf (stderr, "optimize_norm_maximum: start\n");
01104 #endif
01105   e = 0.;
01106   for (i = 0; i < optimize->nexperiments; ++i)
01107     {
01108       ei = fabs (optimize_parse (simulation, i));
01109       e = fmax (e, ei);
01110     }
01111 #if DEBUG
01112   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
01113   fprintf (stderr, "optimize_norm_maximum: end\n");
01114 #endif
01115   return e;
01116 }
01117
01125 double
01126 optimize_norm_p (unsigned int simulation)
01127 {
01128   double e, ei;
01129   unsigned int i;
01130 #if DEBUG
01131   fprintf (stderr, "optimize_norm_p: start\n");
01132 #endif
01133   e = 0.;
01134   for (i = 0; i < optimize->nexperiments; ++i)
01135     {
01136       ei = fabs (optimize_parse (simulation, i));
01137       e += pow (ei, optimize->p);
```

```
01138     }
01139   e = pow (e, 1. / optimize->p);
01140 #if DEBUG
01141   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
01142   fprintf (stderr, "optimize_norm_p: end\n");
01143 #endif
01144   return e;
01145 }
01146
01154 double
01155 optimize_norm_taxicab (unsigned int simulation)
01156 {
01157   double e;
01158   unsigned int i;
01159 #if DEBUG
01160   fprintf (stderr, "optimize_norm_taxicab: start\n");
01161 #endif
01162   e = 0.;
01163   for (i = 0; i < optimize->nexperiments; ++i)
01164     e += fabs (optimize_parse (simulation, i));
01165 #if DEBUG
01166   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
01167   fprintf (stderr, "optimize_norm_taxicab: end\n");
01168 #endif
01169   return e;
01170 }
01171
01176 void
01177 optimize_print ()
01178 {
01179   unsigned int i;
01180   char buffer[512];
01181 #if HAVE_MPI
01182   if (optimize->mpi_rank)
01183     return;
01184 #endif
01185   printf ("%s\n", gettext ("Best result"));
01186   fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
01187   printf ("error = %.15le\n", optimize->error_old[0]);
01188   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
01189   for (i = 0; i < optimize->nvariables; ++i)
01190     {
01191       snprintf (buffer, 512, "%s = %s\n",
01192                 optimize->label[i], format[optimize->precision[i]]);
01193       printf (buffer, optimize->value_old[i]);
01194       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
01195     }
01196   fflush (optimize->file_result);
01197 }
01198
01207 void
01208 optimize_save_variables (unsigned int simulation, double error)
01209 {
01210   unsigned int i;
01211   char buffer[64];
01212 #if DEBUG
01213   fprintf (stderr, "optimize_save_variables: start\n");
01214 #endif
01215   for (i = 0; i < optimize->nvariables; ++i)
01216     {
01217       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
01218       fprintf (optimize->file_variables, buffer,
01219                optimize->value[simulation * optimize->nvariables + i]);
01220     }
01221   fprintf (optimize->file_variables, "%.14le\n", error);
01222 #if DEBUG
01223   fprintf (stderr, "optimize_save_variables: end\n");
01224 #endif
01225 }
01226
01235 void
01236 optimize_best (unsigned int simulation, double value)
01237 {
01238   unsigned int i, j;
01239   double e;
01240 #if DEBUG
01241   fprintf (stderr, "optimize_best: start\n");
01242   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
01243            optimize->nsaveds, optimize->nbest);
01244 #endif
01245   if (optimize->nsaveds < optimize->nbest
01246       || value < optimize->error_best[optimize->nsaveds - 1])
01247     {
01248       if (optimize->nsaveds < optimize->nbest)
01249         ++optimize->nsaveds;
01250       optimize->error_best[optimize->nsaveds - 1] = value;
```

```
01251       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
01252       for (i = optimize->nsaveds; --i;)
01253         {
01254           if (optimize->error_best[i] < optimize->error_best[i - 1])
01255             {
01256               j = optimize->simulation_best[i];
01257               e = optimize->error_best[i];
01258               optimize->simulation_best[i] = optimize->
    simulation_best[i - 1];
01259               optimize->error_best[i] = optimize->error_best[i - 1];
01260               optimize->simulation_best[i - 1] = j;
01261               optimize->error_best[i - 1] = e;
01262             }
01263           else
01264             break;
01265         }
01266     }
01267 #if DEBUG
01268   fprintf (stderr, "optimize_best: end\n");
01269 #endif
01270 }
01271
01276 void
01277 optimize_sequential ()
01278 {
01279   unsigned int i;
01280   double e;
01281 #if DEBUG
01282   fprintf (stderr, "optimize_sequential: start\n");
01283   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
01284            optimize->nstart, optimize->nend);
01285 #endif
01286   for (i = optimize->nstart; i < optimize->nend; ++i)
01287     {
01288       e = optimize_norm (i);
01289       optimize_best (i, e);
01290       optimize_save_variables (i, e);
01291       if (e < optimize->thresold)
01292         {
01293           optimize->stop = 1;
01294           break;
01295         }
01296 #if DEBUG
01297       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
01298 #endif
01299     }
01300 #if DEBUG
01301   fprintf (stderr, "optimize_sequential: end\n");
01302 #endif
01303 }
01304
01312 void *
01313 optimize_thread (ParallelData * data)
01314 {
01315   unsigned int i, thread;
01316   double e;
01317 #if DEBUG
01318   fprintf (stderr, "optimize_thread: start\n");
01319 #endif
01320   thread = data->thread;
01321 #if DEBUG
01322   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
01323            optimize->thread[thread], optimize->thread[thread + 1]);
01324 #endif
01325   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
01326     {
01327       e = optimize_norm (i);
01328       g_mutex_lock (mutex);
01329       optimize_best (i, e);
01330       optimize_save_variables (i, e);
01331       if (e < optimize->thresold)
01332         optimize->stop = 1;
01333       g_mutex_unlock (mutex);
01334       if (optimize->stop)
01335         break;
01336 #if DEBUG
01337       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
01338 #endif
01339     }
01340 #if DEBUG
01341   fprintf (stderr, "optimize_thread: end\n");
01342 #endif
01343   g_thread_exit (NULL);
01344   return NULL;
01345 }
01346
01358 void
```

```
01359 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
01360                 double *error_best)
01361 {
01362   unsigned int i, j, k, s[optimize->nbest];
01363   double e[optimize->nbest];
01364 #if DEBUG
01365   fprintf (stderr, "optimize_merge: start\n");
01366 #endif
01367   i = j = k = 0;
01368   do
01369     {
01370       if (i == optimize->nsaveds)
01371         {
01372           s[k] = simulation_best[j];
01373           e[k] = error_best[j];
01374           ++j;
01375           ++k;
01376           if (j == nsaveds)
01377             break;
01378         }
01379       else if (j == nsaveds)
01380         {
01381           s[k] = optimize->simulation_best[i];
01382           e[k] = optimize->error_best[i];
01383           ++i;
01384           ++k;
01385           if (i == optimize->nsaveds)
01386             break;
01387         }
01388       else if (optimize->error_best[i] > error_best[j])
01389         {
01390           s[k] = simulation_best[j];
01391           e[k] = error_best[j];
01392           ++j;
01393           ++k;
01394         }
01395       else
01396         {
01397           s[k] = optimize->simulation_best[i];
01398           e[k] = optimize->error_best[i];
01399           ++i;
01400           ++k;
01401         }
01402     }
01403   while (k < optimize->nbest);
01404   optimize->nsaveds = k;
01405   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
01406   memcpy (optimize->error_best, e, k * sizeof (double));
01407 #if DEBUG
01408   fprintf (stderr, "optimize_merge: end\n");
01409 #endif
01410 }
01411
01416 #if HAVE_MPI
01417 void
01418 optimize_synchronise ()
01419 {
01420   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
01421   double error_best[optimize->nbest];
01422   MPI_Status mpi_stat;
01423 #if DEBUG
01424   fprintf (stderr, "optimize_synchronise: start\n");
01425 #endif
01426   if (optimize->mpi_rank == 0)
01427     {
01428       for (i = 1; i < ntasks; ++i)
01429         {
01430           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01431           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01432                     MPI_COMM_WORLD, &mpi_stat);
01433           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01434                     MPI_COMM_WORLD, &mpi_stat);
01435           optimize_merge (nsaveds, simulation_best, error_best);
01436           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
01437           if (stop)
01438             optimize->stop = 1;
01439         }
01440       for (i = 1; i < ntasks; ++i)
01441         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
01442     }
01443   else
01444     {
01445       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01446       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
01447                 MPI_COMM_WORLD);
01448       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
01449                 MPI_COMM_WORLD);
```

```
01450       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
01451       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
01452       if (stop)
01453         optimize->stop = 1;
01454     }
01455 #if DEBUG
01456   fprintf (stderr, "optimize_synchronise: end\n");
01457 #endif
01458 }
01459 #endif
01460
01465 void
01466 optimize_sweep ()
01467 {
01468   unsigned int i, j, k, l;
01469   double e;
01470   GThread *thread[nthreads];
01471   ParallelData data[nthreads];
01472 #if DEBUG
01473   fprintf (stderr, "optimize_sweep: start\n");
01474 #endif
01475   for (i = 0; i < optimize->nsimulations; ++i)
01476     {
01477       k = i;
01478       for (j = 0; j < optimize->nvariables; ++j)
01479         {
01480           l = k % optimize->nsweeps[j];
01481           k /= optimize->nsweeps[j];
01482           e = optimize->rangemin[j];
01483           if (optimize->nsweeps[j] > 1)
01484             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
01485               / (optimize->nsweeps[j] - 1);
01486           optimize->value[i * optimize->nvariables + j] = e;
01487         }
01488     }
01489   optimize->nsaveds = 0;
01490   if (nthreads <= 1)
01491     optimize_sequential ();
01492   else
01493     {
01494       for (i = 0; i < nthreads; ++i)
01495         {
01496           data[i].thread = i;
01497           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
01498         }
01499       for (i = 0; i < nthreads; ++i)
01500         g_thread_join (thread[i]);
01501     }
01502 #if HAVE_MPI
01503   // Communicating tasks results
01504   optimize_synchronise ();
01505 #endif
01506 #if DEBUG
01507   fprintf (stderr, "optimize_sweep: end\n");
01508 #endif
01509 }
01510
01515 void
01516 optimize_MonteCarlo ()
01517 {
01518   unsigned int i, j;
01519   GThread *thread[nthreads];
01520   ParallelData data[nthreads];
01521 #if DEBUG
01522   fprintf (stderr, "optimize_MonteCarlo: start\n");
01523 #endif
01524   for (i = 0; i < optimize->nsimulations; ++i)
01525     for (j = 0; j < optimize->nvariables; ++j)
01526       optimize->value[i * optimize->nvariables + j]
01527         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
01528         * (optimize->rangemax[j] - optimize->rangemin[j]);
01529   optimize->nsaveds = 0;
01530   if (nthreads <= 1)
01531     optimize_sequential ();
01532   else
01533     {
01534       for (i = 0; i < nthreads; ++i)
01535         {
01536           data[i].thread = i;
01537           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
01538         }
01539       for (i = 0; i < nthreads; ++i)
01540         g_thread_join (thread[i]);
01541     }
01542 #if HAVE_MPI
01543   // Communicating tasks results
01544   optimize_synchronise ();
```

```
01545 #endif
01546 #if DEBUG
01547   fprintf (stderr, "optimize_MonteCarlo: end\n");
01548 #endif
01549 }
01550
01560 void
01561 optimize_best_direction (unsigned int simulation, double value)
01562 {
01563 #if DEBUG
01564   fprintf (stderr, "optimize_best_direction: start\n");
01565   fprintf (stderr,
01566            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
01567            simulation, value, optimize->error_best[0]);
01568 #endif
01569   if (value < optimize->error_best[0])
01570     {
01571       optimize->error_best[0] = value;
01572       optimize->simulation_best[0] = simulation;
01573 #if DEBUG
01574       fprintf (stderr,
01575                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
01576                simulation, value);
01577 #endif
01578     }
01579 #if DEBUG
01580   fprintf (stderr, "optimize_best_direction: end\n");
01581 #endif
01582 }
01583
01590 void
01591 optimize_direction_sequential (unsigned int simulation)
01592 {
01593   unsigned int i, j;
01594   double e;
01595 #if DEBUG
01596   fprintf (stderr, "optimize_direction_sequential: start\n");
01597   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
01598            "nend_direction=%u\n",
01599            optimize->nstart_direction, optimize->nend_direction);
01600 #endif
01601   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
01602     {
01603       j = simulation + i;
01604       e = optimize_norm (j);
01605       optimize_best_direction (j, e);
01606       optimize_save_variables (j, e);
01607       if (e < optimize->thresold)
01608         {
01609           optimize->stop = 1;
01610           break;
01611         }
01612 #if DEBUG
01613       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
01614 #endif
01615     }
01616 #if DEBUG
01617   fprintf (stderr, "optimize_direction_sequential: end\n");
01618 #endif
01619 }
01620
01628 void *
01629 optimize_direction_thread (ParallelData * data)
01630 {
01631   unsigned int i, thread;
01632   double e;
01633 #if DEBUG
01634   fprintf (stderr, "optimize_direction_thread: start\n");
01635 #endif
01636   thread = data->thread;
01637 #if DEBUG
01638   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
01639            thread,
01640            optimize->thread_direction[thread],
01641            optimize->thread_direction[thread + 1]);
01642 #endif
01643   for (i = optimize->thread_direction[thread];
01644        i < optimize->thread_direction[thread + 1]; ++i)
01645     {
01646       e = optimize_norm (i);
01647       g_mutex_lock (mutex);
01648       optimize_best_direction (i, e);
01649       optimize_save_variables (i, e);
01650       if (e < optimize->thresold)
01651         optimize->stop = 1;
01652       g_mutex_unlock (mutex);
01653       if (optimize->stop)
```

```
01654         break;
01655 #if DEBUG
01656       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
01657 #endif
01658     }
01659 #if DEBUG
01660   fprintf (stderr, "optimize_direction_thread: end\n");
01661 #endif
01662   g_thread_exit (NULL);
01663   return NULL;
01664 }
01665
01675 double
01676 optimize_estimate_direction_random (unsigned int variable,
01677                                     unsigned int estimate)
01678 {
01679   double x;
01680 #if DEBUG
01681   fprintf (stderr, "optimize_estimate_direction_random: start\n");
01682 #endif
01683   x = optimize->direction[variable]
01684     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
01685 #if DEBUG
01686   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
01687            variable, x);
01688   fprintf (stderr, "optimize_estimate_direction_random: end\n");
01689 #endif
01690   return x;
01691 }
01692
01702 double
01703 optimize_estimate_direction_coordinates (unsigned int variable,
01704                                          unsigned int estimate)
01705 {
01706   double x;
01707 #if DEBUG
01708   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
01709 #endif
01710   x = optimize->direction[variable];
01711   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01712     {
01713       if (estimate & 1)
01714         x += optimize->step[variable];
01715       else
01716         x -= optimize->step[variable];
01717     }
01718 #if DEBUG
01719   fprintf (stderr,
01720            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
01721            variable, x);
01722   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
01723 #endif
01724   return x;
01725 }
01726
01733 void
01734 optimize_step_direction (unsigned int simulation)
01735 {
01736   GThread *thread[nthreads_direction];
01737   ParallelData data[nthreads_direction];
01738   unsigned int i, j, k, b;
01739 #if DEBUG
01740   fprintf (stderr, "optimize_step_direction: start\n");
01741 #endif
01742   for (i = 0; i < optimize->nestimates; ++i)
01743     {
01744       k = (simulation + i) * optimize->nvariables;
01745       b = optimize->simulation_best[0] * optimize->nvariables;
01746 #if DEBUG
01747       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
01748                simulation + i, optimize->simulation_best[0]);
01749 #endif
01750       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
01751         {
01752 #if DEBUG
01753           fprintf (stderr,
01754                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
01755                    i, j, optimize->value[b]);
01756 #endif
01757           optimize->value[k]
01758             = optimize->value[b] + optimize_estimate_direction (j, i);
01759           optimize->value[k] = fmin (fmax (optimize->value[k],
01760                                            optimize->rangeminabs[j]),
01761                                      optimize->rangemaxabs[j]);
01762 #if DEBUG
01763           fprintf (stderr,
01764                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
```

```
01765                   i, j, optimize->value[k]);
01766 #endif
01767         }
01768     }
01769   if (nthreads_direction == 1)
01770     optimize_direction_sequential (simulation);
01771   else
01772     {
01773       for (i = 0; i <= nthreads_direction; ++i)
01774         {
01775           optimize->thread_direction[i]
01776             = simulation + optimize->nstart_direction
01777             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01778             / nthreads_direction;
01779 #if DEBUG
01780           fprintf (stderr,
01781                    "optimize_step_direction: i=%u thread_direction=%u\n",
01782                    i, optimize->thread_direction[i]);
01783 #endif
01784         }
01785       for (i = 0; i < nthreads_direction; ++i)
01786         {
01787           data[i].thread = i;
01788           thread[i] = g_thread_new
01789             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01790         }
01791       for (i = 0; i < nthreads_direction; ++i)
01792         g_thread_join (thread[i]);
01793     }
01794 #if DEBUG
01795   fprintf (stderr, "optimize_step_direction: end\n");
01796 #endif
01797 }
01798
01803 void
01804 optimize_direction ()
01805 {
01806   unsigned int i, j, k, b, s, adjust;
01807 #if DEBUG
01808   fprintf (stderr, "optimize_direction: start\n");
01809 #endif
01810   for (i = 0; i < optimize->nvariables; ++i)
01811     optimize->direction[i] = 0.;
01812   b = optimize->simulation_best[0] * optimize->nvariables;
01813   s = optimize->nsimulations;
01814   adjust = 1;
01815   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01816     {
01817 #if DEBUG
01818       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01819                i, optimize->simulation_best[0]);
01820 #endif
01821       optimize_step_direction (s);
01822       k = optimize->simulation_best[0] * optimize->nvariables;
01823 #if DEBUG
01824       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01825                i, optimize->simulation_best[0]);
01826 #endif
01827       if (k == b)
01828         {
01829           if (adjust)
01830             for (j = 0; j < optimize->nvariables; ++j)
01831               optimize->step[j] *= 0.5;
01832           for (j = 0; j < optimize->nvariables; ++j)
01833             optimize->direction[j] = 0.;
01834           adjust = 1;
01835         }
01836       else
01837         {
01838           for (j = 0; j < optimize->nvariables; ++j)
01839             {
01840 #if DEBUG
01841               fprintf (stderr,
01842                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01843                        j, optimize->value[k + j], j, optimize->value[b + j]);
01844 #endif
01845               optimize->direction[j]
01846                 = (1. - optimize->relaxation) * optimize->direction[j]
01847                 + optimize->relaxation
01848                 * (optimize->value[k + j] - optimize->value[b + j]);
01849 #if DEBUG
01850               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01851                        j, optimize->direction[j]);
01852 #endif
01853             }
01854           adjust = 0;
```

```
01855         }
01856     }
01857 #if DEBUG
01858   fprintf (stderr, "optimize_direction: end\n");
01859 #endif
01860 }
01861
01869 double
01870 optimize_genetic_objective (Entity * entity)
01871 {
01872   unsigned int j;
01873   double objective;
01874   char buffer[64];
01875 #if DEBUG
01876   fprintf (stderr, "optimize_genetic_objective: start\n");
01877 #endif
01878   for (j = 0; j < optimize->nvariables; ++j)
01879     {
01880       optimize->value[entity->id * optimize->nvariables + j]
01881         = genetic_get_variable (entity, optimize->genetic_variable + j);
01882     }
01883   objective = optimize_norm (entity->id);
01884   g_mutex_lock (mutex);
01885   for (j = 0; j < optimize->nvariables; ++j)
01886     {
01887       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01888       fprintf (optimize->file_variables, buffer,
01889                genetic_get_variable (entity, optimize->genetic_variable + j));
01890     }
01891   fprintf (optimize->file_variables, "%.14le\n", objective);
01892   g_mutex_unlock (mutex);
01893 #if DEBUG
01894   fprintf (stderr, "optimize_genetic_objective: end\n");
01895 #endif
01896   return objective;
01897 }
01898
01903 void
01904 optimize_genetic ()
01905 {
01906   char *best_genome;
01907   double best_objective, *best_variable;
01908 #if DEBUG
01909   fprintf (stderr, "optimize_genetic: start\n");
01910   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01911            nthreads);
01912   fprintf (stderr,
01913            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01914            optimize->nvariables, optimize->nsimulations, optimize->
      niterations);
01915   fprintf (stderr,
01916            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01917            optimize->mutation_ratio, optimize->reproduction_ratio,
01918            optimize->adaptation_ratio);
01919 #endif
01920   genetic_algorithm_default (optimize->nvariables,
01921                              optimize->genetic_variable,
01922                              optimize->nsimulations,
01923                              optimize->niterations,
01924                              optimize->mutation_ratio,
01925                              optimize->reproduction_ratio,
01926                              optimize->adaptation_ratio,
01927                              optimize->thresold,
01928                              &optimize_genetic_objective,
01929                              &best_genome, &best_variable, &best_objective);
01930 #if DEBUG
01931   fprintf (stderr, "optimize_genetic: the best\n");
01932 #endif
01933   optimize->error_old = (double *) g_malloc (sizeof (double));
01934   optimize->value_old
01935     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01936   optimize->error_old[0] = best_objective;
01937   memcpy (optimize->value_old, best_variable,
01938           optimize->nvariables * sizeof (double));
01939   g_free (best_genome);
01940   g_free (best_variable);
01941   optimize_print ();
01942 #if DEBUG
01943   fprintf (stderr, "optimize_genetic: end\n");
01944 #endif
01945 }
01946
01951 void
01952 optimize_save_old ()
01953 {
01954   unsigned int i, j;
01955 #if DEBUG
```

```
01956   fprintf (stderr, "optimize_save_old: start\n");
01957   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01958 #endif
01959   memcpy (optimize->error_old, optimize->error_best,
01960           optimize->nbest * sizeof (double));
01961   for (i = 0; i < optimize->nbest; ++i)
01962     {
01963       j = optimize->simulation_best[i];
01964 #if DEBUG
01965       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01966 #endif
01967       memcpy (optimize->value_old + i * optimize->nvariables,
01968               optimize->value + j * optimize->nvariables,
01969               optimize->nvariables * sizeof (double));
01970     }
01971 #if DEBUG
01972   for (i = 0; i < optimize->nvariables; ++i)
01973     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01974              i, optimize->value_old[i]);
01975   fprintf (stderr, "optimize_save_old: end\n");
01976 #endif
01977 }
01978
01984 void
01985 optimize_merge_old ()
01986 {
01987   unsigned int i, j, k;
01988   double v[optimize->nbest * optimize->nvariables], e[optimize->
      nbest],
01989     *enew, *eold;
01990 #if DEBUG
01991   fprintf (stderr, "optimize_merge_old: start\n");
01992 #endif
01993   enew = optimize->error_best;
01994   eold = optimize->error_old;
01995   i = j = k = 0;
01996   do
01997     {
01998       if (*enew < *eold)
01999         {
02000           memcpy (v + k * optimize->nvariables,
02001                   optimize->value
02002                   + optimize->simulation_best[i] * optimize->
      nvariables,
02003                   optimize->nvariables * sizeof (double));
02004           e[k] = *enew;
02005           ++k;
02006           ++enew;
02007           ++i;
02008         }
02009       else
02010         {
02011           memcpy (v + k * optimize->nvariables,
02012                   optimize->value_old + j * optimize->nvariables,
02013                   optimize->nvariables * sizeof (double));
02014           e[k] = *eold;
02015           ++k;
02016           ++eold;
02017           ++j;
02018         }
02019     }
02020   while (k < optimize->nbest);
02021   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
02022   memcpy (optimize->error_old, e, k * sizeof (double));
02023 #if DEBUG
02024   fprintf (stderr, "optimize_merge_old: end\n");
02025 #endif
02026 }
02027
02033 void
02034 optimize_refine ()
02035 {
02036   unsigned int i, j;
02037   double d;
02038 #if HAVE_MPI
02039   MPI_Status mpi_stat;
02040 #endif
02041 #if DEBUG
02042   fprintf (stderr, "optimize_refine: start\n");
02043 #endif
02044 #if HAVE_MPI
02045   if (!optimize->mpi_rank)
02046     {
02047 #endif
02048       for (j = 0; j < optimize->nvariables; ++j)
02049         {
02050           optimize->rangemin[j] = optimize->rangemax[j]
```

```
02051             = optimize->value_old[j];
02052         }
02053       for (i = 0; ++i < optimize->nbest;)
02054         {
02055           for (j = 0; j < optimize->nvariables; ++j)
02056             {
02057               optimize->rangemin[j]
02058                 = fmin (optimize->rangemin[j],
02059                         optimize->value_old[i * optimize->nvariables + j]);
02060               optimize->rangemax[j]
02061                 = fmax (optimize->rangemax[j],
02062                         optimize->value_old[i * optimize->nvariables + j]);
02063             }
02064         }
02065       for (j = 0; j < optimize->nvariables; ++j)
02066         {
02067           d = optimize->tolerance
02068             * (optimize->rangemax[j] - optimize->rangemin[j]);
02069           switch (optimize->algorithm)
02070             {
02071             case ALGORITHM_MONTE_CARLO:
02072               d *= 0.5;
02073               break;
02074             default:
02075               if (optimize->nsweeps[j] > 1)
02076                 d /= optimize->nsweeps[j] - 1;
02077               else
02078                 d = 0.;
02079             }
02080           optimize->rangemin[j] -= d;
02081           optimize->rangemin[j]
02082             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
02083           optimize->rangemax[j] += d;
02084           optimize->rangemax[j]
02085             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
02086           printf ("%s min=%lg max=%lg\n", optimize->label[j],
02087                   optimize->rangemin[j], optimize->rangemax[j]);
02088           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
02089                    optimize->label[j], optimize->rangemin[j],
02090                    optimize->rangemax[j]);
02091         }
02092 #if HAVE_MPI
02093       for (i = 1; i < ntasks; ++i)
02094         {
02095           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
02096                     1, MPI_COMM_WORLD);
02097           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
02098                     1, MPI_COMM_WORLD);
02099         }
02100     }
02101   else
02102     {
02103       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
02104                 MPI_COMM_WORLD, &mpi_stat);
02105       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
02106                 MPI_COMM_WORLD, &mpi_stat);
02107     }
02108 #endif
02109 #if DEBUG
02110   fprintf (stderr, "optimize_refine: end\n");
02111 #endif
02112 }
02113
02118 void
02119 optimize_step ()
02120 {
02121 #if DEBUG
02122   fprintf (stderr, "optimize_step: start\n");
02123 #endif
02124   optimize_algorithm ();
02125   if (optimize->nsteps)
02126     optimize_direction ();
02127 #if DEBUG
02128   fprintf (stderr, "optimize_step: end\n");
02129 #endif
02130 }
02131
02136 void
02137 optimize_iterate ()
02138 {
02139   unsigned int i;
02140 #if DEBUG
02141   fprintf (stderr, "optimize_iterate: start\n");
02142 #endif
02143   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
02144   optimize->value_old = (double *)
02145     g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
```

```
02146   optimize_step ();
02147   optimize_save_old ();
02148   optimize_refine ();
02149   optimize_print ();
02150   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
02151     {
02152       optimize_step ();
02153       optimize_merge_old ();
02154       optimize_refine ();
02155       optimize_print ();
02156     }
02157 #if DEBUG
02158   fprintf (stderr, "optimize_iterate: end\n");
02159 #endif
02160 }
02161
02166 void
02167 optimize_free ()
02168 {
02169   unsigned int i, j;
02170 #if DEBUG
02171   fprintf (stderr, "optimize_free: start\n");
02172 #endif
02173   for (j = 0; j < optimize->ninputs; ++j)
02174     {
02175       for (i = 0; i < optimize->nexperiments; ++i)
02176         g_mapped_file_unref (optimize->file[j][i]);
02177       g_free (optimize->file[j]);
02178     }
02179   g_free (optimize->error_old);
02180   g_free (optimize->value_old);
02181   g_free (optimize->value);
02182   g_free (optimize->genetic_variable);
02183   g_free (optimize->rangemax);
02184   g_free (optimize->rangemin);
02185 #if DEBUG
02186   fprintf (stderr, "optimize_free: end\n");
02187 #endif
02188 }
02189
02194 void
02195 optimize_open ()
02196 {
02197   GTimeZone *tz;
02198   GDateTime *t0, *t;
02199   unsigned int i, j, *nbits;
02200
02201 #if DEBUG
02202   char *buffer;
02203   fprintf (stderr, "optimize_open: start\n");
02204 #endif
02205
02206   // Getting initial time
02207 #if DEBUG
02208   fprintf (stderr, "optimize_open: getting initial time\n");
02209 #endif
02210   tz = g_time_zone_new_utc ();
02211   t0 = g_date_time_new_now (tz);
02212
02213   // Obtaining and initing the pseudo-random numbers generator seed
02214 #if DEBUG
02215   fprintf (stderr, "optimize_open: getting initial seed\n");
02216 #endif
02217   optimize->seed = input->seed;
02218   gsl_rng_set (optimize->rng, optimize->seed);
02219
02220   // Replacing the working directory
02221 #if DEBUG
02222   fprintf (stderr, "optimize_open: replacing the working directory\n");
02223 #endif
02224   g_chdir (input->directory);
02225
02226   // Getting results file names
02227   optimize->result = input->result;
02228   optimize->variables = input->variables;
02229
02230   // Obtaining the simulator file
02231   optimize->simulator = input->simulator;
02232
02233   // Obtaining the evaluator file
02234   optimize->evaluator = input->evaluator;
02235
02236   // Reading the algorithm
02237   optimize->algorithm = input->algorithm;
02238   switch (optimize->algorithm)
02239     {
02240     case ALGORITHM_MONTE_CARLO:
```

```
02241       optimize_algorithm = optimize_MonteCarlo;
02242       break;
02243     case ALGORITHM_SWEEP:
02244       optimize_algorithm = optimize_sweep;
02245       break;
02246     default:
02247       optimize_algorithm = optimize_genetic;
02248       optimize->mutation_ratio = input->mutation_ratio;
02249       optimize->reproduction_ratio = input->reproduction_ratio;
02250       optimize->adaptation_ratio = input->adaptation_ratio;
02251     }
02252   optimize->nvariables = input->nvariables;
02253   optimize->nsimulations = input->nsimulations;
02254   optimize->niterations = input->niterations;
02255   optimize->nbest = input->nbest;
02256   optimize->tolerance = input->tolerance;
02257   optimize->nsteps = input->nsteps;
02258   optimize->nestimates = 0;
02259   optimize->thresold = input->thresold;
02260   optimize->stop = 0;
02261   if (input->nsteps)
02262     {
02263       optimize->relaxation = input->relaxation;
02264       switch (input->direction)
02265         {
02266         case DIRECTION_METHOD_COORDINATES:
02267           optimize->nestimates = 2 * optimize->nvariables;
02268           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
02269           break;
02270         default:
02271           optimize->nestimates = input->nestimates;
02272           optimize_estimate_direction =
      optimize_estimate_direction_random;
02273         }
02274     }
02275
02276 #if DEBUG
02277   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
02278 #endif
02279   optimize->simulation_best
02280     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
02281   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
02282
02283   // Reading the experimental data
02284 #if DEBUG
02285   buffer = g_get_current_dir ();
02286   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
02287   g_free (buffer);
02288 #endif
02289   optimize->nexperiments = input->nexperiments;
02290   optimize->ninputs = input->ninputs;
02291   optimize->experiment = input->experiment;
02292   optimize->weight = input->weight;
02293   for (i = 0; i < input->ninputs; ++i)
02294     {
02295       optimize->template[i] = input->template[i];
02296       optimize->file[i]
02297         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02298     }
02299   for (i = 0; i < input->nexperiments; ++i)
02300     {
02301 #if DEBUG
02302       fprintf (stderr, "optimize_open: i=%u\n", i);
02303       fprintf (stderr, "optimize_open: experiment=%s\n",
02304                optimize->experiment[i]);
02305       fprintf (stderr, "optimize_open: weight=%lg\n", optimize->weight[i]);
02306 #endif
02307       for (j = 0; j < input->ninputs; ++j)
02308         {
02309 #if DEBUG
02310           fprintf (stderr, "optimize_open: template%u\n", j + 1);
02311           fprintf (stderr, "optimize_open: experiment=%u template%u=%s\n",
02312                    i, j + 1, optimize->template[j][i]);
02313 #endif
02314           optimize->file[j][i]
02315             = g_mapped_file_new (input->template[j][i], 0, NULL);
02316         }
02317     }
02318
02319   // Reading the variables data
02320 #if DEBUG
02321   fprintf (stderr, "optimize_open: reading variables\n");
02322 #endif
02323   optimize->label = input->label;
02324   j = input->nvariables * sizeof (double);
02325   optimize->rangemin = (double *) g_malloc (j);
```

```
02326   optimize->rangemax = (double *) g_malloc (j);
02327   memcpy (optimize->rangemin, input->rangemin, j);
02328   memcpy (optimize->rangemax, input->rangemax, j);
02329   optimize->rangeminabs = input->rangeminabs;
02330   optimize->rangemaxabs = input->rangemaxabs;
02331   optimize->precision = input->precision;
02332   optimize->nsweeps = input->nsweeps;
02333   optimize->step = input->step;
02334   nbits = input->nbits;
02335   if (input->algorithm == ALGORITHM_SWEEP)
02336     {
02337       optimize->nsimulations = 1;
02338       for (i = 0; i < input->nvariables; ++i)
02339         {
02340           if (input->algorithm == ALGORITHM_SWEEP)
02341             {
02342               optimize->nsimulations *= input->nsweeps[i];
02343 #if DEBUG
02344               fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
02345                        optimize->nsweeps[i], optimize->nsimulations);
02346 #endif
02347             }
02348         }
02349     }
02350   if (optimize->nsteps)
02351     optimize->direction
02352       = (double *) alloca (optimize->nvariables * sizeof (double));
02353
02354   // Setting error norm
02355   switch (input->norm)
02356     {
02357     case ERROR_NORM_EUCLIDIAN:
02358       optimize_norm = optimize_norm_euclidian;
02359       break;
02360     case ERROR_NORM_MAXIMUM:
02361       optimize_norm = optimize_norm_maximum;
02362       break;
02363     case ERROR_NORM_P:
02364       optimize_norm = optimize_norm_p;
02365       optimize->p = input->p;
02366       break;
02367     default:
02368       optimize_norm = optimize_norm_taxicab;
02369     }
02370
02371   // Allocating values
02372 #if DEBUG
02373   fprintf (stderr, "optimize_open: allocating variables\n");
02374   fprintf (stderr, "optimize_open: nvariables=%u\n", optimize->nvariables);
02375 #endif
02376   optimize->genetic_variable = NULL;
02377   if (optimize->algorithm == ALGORITHM_GENETIC)
02378     {
02379       optimize->genetic_variable = (GeneticVariable *)
02380         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
02381       for (i = 0; i < optimize->nvariables; ++i)
02382         {
02383 #if DEBUG
02384           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
02385                    i, optimize->rangemin[i], optimize->rangemax[i], nbits[i]);
02386 #endif
02387           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
02388           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
02389           optimize->genetic_variable[i].nbits = nbits[i];
02390         }
02391     }
02392 #if DEBUG
02393   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
02394            optimize->nvariables, optimize->nsimulations);
02395 #endif
02396   optimize->value = (double *)
02397     g_malloc ((optimize->nsimulations
02398                + optimize->nestimates * optimize->nsteps)
02399               * optimize->nvariables * sizeof (double));
02400
02401   // Calculating simulations to perform for each task
02402 #if HAVE_MPI
02403 #if DEBUG
02404   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
02405            optimize->mpi_rank, ntasks);
02406 #endif
02407   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
02408   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
```

```
02409   if (optimize->nsteps)
02410     {
02411       optimize->nstart_direction
02412         = optimize->mpi_rank * optimize->nestimates / ntasks;
02413       optimize->nend_direction
02414         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
02415     }
02416 #else
02417   optimize->nstart = 0;
02418   optimize->nend = optimize->nsimulations;
02419   if (optimize->nsteps)
02420     {
02421       optimize->nstart_direction = 0;
02422       optimize->nend_direction = optimize->nestimates;
02423     }
02424 #endif
02425 #if DEBUG
02426   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
02427            optimize->nend);
02428 #endif
02429
02430   // Calculating simulations to perform for each thread
02431   optimize->thread
02432     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02433   for (i = 0; i <= nthreads; ++i)
02434     {
02435       optimize->thread[i] = optimize->nstart
02436         + i * (optimize->nend - optimize->nstart) / nthreads;
02437 #if DEBUG
02438       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
02439                optimize->thread[i]);
02440 #endif
02441     }
02442   if (optimize->nsteps)
02443     optimize->thread_direction = (unsigned int *)
02444       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
02445
02446   // Opening result files
02447   optimize->file_result = g_fopen (optimize->result, "w");
02448   optimize->file_variables = g_fopen (optimize->variables, "w");
02449
02450   // Performing the algorithm
02451   switch (optimize->algorithm)
02452     {
02453       // Genetic algorithm
02454     case ALGORITHM_GENETIC:
02455       optimize_genetic ();
02456       break;
02457
02458       // Iterative algorithm
02459     default:
02460       optimize_iterate ();
02461     }
02462
02463   // Getting calculation time
02464   t = g_date_time_new_now (tz);
02465   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02466   g_date_time_unref (t);
02467   g_date_time_unref (t0);
02468   g_time_zone_unref (tz);
02469   printf ("%s = %.6lg s\n",
02470           gettext ("Calculation time"), optimize->calculation_time);
02471   fprintf (optimize->file_result, "%s = %.6lg s\n",
02472            gettext ("Calculation time"), optimize->calculation_time);
02473
02474   // Closing result files
02475   fclose (optimize->file_variables);
02476   fclose (optimize->file_result);
02477
02478 #if DEBUG
02479   fprintf (stderr, "optimize_open: end\n");
02480 #endif
02481 }
```

5.11 optimize.h File Reference

Header file to define the optimization functions.

This graph shows which files directly or indirectly include this file:

Data Structures

- struct Input

  Struct to define the optimization input file.
- struct Optimize

  Struct to define the optimization ation data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

- enum Algorithm { ALGORITHM_MONTE_CARLO = 0, ALGORITHM_SWEEP = 1, ALGORITHM_GENETIC = 2 }

  Enum to define the algorithms.
- enum DirectionMethod { DIRECTION_METHOD_COORDINATES = 0, DIRECTION_METHOD_RANDOM = 1 }

  Enum to define the methods to estimate the direction search.
- enum ErrorNorm { ERROR_NORM_EUCLIDIAN = 0, ERROR_NORM_MAXIMUM = 1, ERROR_NORM_P = 2, ERROR_NORM_TAXICAB = 3 }

  Enum to define the error norm.

Functions

- void input_new ()

  Function to create a new Input struct.
- void input_free ()

  Function to free the memory of the input file data.
- int input_open (char *filename)

  Function to open the input file.
- void optimize_input (unsigned int simulation, char *input, GMappedFile *template)

  Function to write the simulation input file.
- double optimize_parse (unsigned int simulation, unsigned int experiment)

  Function to parse input files, simulating and calculating the \ objective function.
- double optimize_norm_euclidian (unsigned int simulation)

  Function to calculate the Euclidian error norm.
- double optimize_norm_maximum (unsigned int simulation)

  Function to calculate the maximum error norm.
- double optimize_norm_p (unsigned int simulation)

  Function to calculate the P error norm.
- double optimize_norm_taxicab (unsigned int simulation)

  Function to calculate the taxicab error norm.
- void optimize_print ()

  Function to print the results.
- void optimize_save_variables (unsigned int simulation, double error)

  Function to save in a file the variables and the error.
- void optimize_best (unsigned int simulation, double value)

  Function to save the best simulations.
- void optimize_sequential ()

  Function to optimize sequentially.
- void * optimize_thread (ParallelData *data)

  Function to optimize on a thread.

- void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 optimization results.
- void optimize_synchronise ()

  Function to synchronise the optimization results of MPI tasks.
- void optimize_sweep ()

  Function to optimize with the sweep algorithm.
- void optimize_MonteCarlo ()

  Function to optimize with the Monte-Carlo algorithm.
- void optimize_best_direction (unsigned int simulation, double value)

  Function to save the best simulation in a direction search method.
- void **optimize_direction_sequential** ()
- void * optimize_direction_thread (ParallelData *data)

  Function to estimate the direction search on a thread.
- double optimize_estimate_direction_random (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- void optimize_step_direction (unsigned int simulation)

  Function to do a step of the direction search method.
- void optimize_direction ()

  Function to optimize with a direction search method.
- double optimize_genetic_objective (Entity *entity)

  Function to calculate the objective function of an entity.
- void optimize_genetic ()

  Function to optimize with the genetic algorithm.
- void optimize_save_old ()

  Function to save the best results on iterative methods.
- void optimize_merge_old ()

  Function to merge the best results with the previous step best results on iterative methods.
- void optimize_refine ()

  Function to refine the search ranges of the variables in iterative algorithms.
- void optimize_step ()

  Function to do a step of the iterative algorithm.
- void optimize_iterate ()

  Function to iterate the algorithm.
- void optimize_free ()

  Function to free the memory used by the Optimize struct.
- void optimize_open ()

  Function to open and perform a optimization.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- unsigned int nthreads_direction

  Number of threads for the direction search method.
- GMutex mutex [1]

  Mutex struct.

- void(∗ optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(∗ optimize_estimate_direction)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the direction.
- double(∗ optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Input input [1]

  Input struct to define the input file to mpcotool.
- Optimize optimize [1]

  Optimization data.
- const xmlChar ∗ result_name

  Name of the result file.
- const xmlChar ∗ variables_name

  Name of the variables file.
- const xmlChar ∗ template [MAX_NINPUTS]

  Array of xmlChar strings with template labels.
- const char ∗ format [NPRECISIONS]

  Array of C-strings with variable formats.
- const double precision [NPRECISIONS]

  Array of variable precisions.

5.11.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file optimize.h.

5.11.2 Enumeration Type Documentation

5.11.2.1 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

Definition at line 45 of file optimize.h.

```
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
```

5.11.2.2 enum DirectionMethod

Enum to define the methods to estimate the direction search.

Enumerator

DIRECTION_METHOD_COORDINATES Coordinates descent method.

DIRECTION_METHOD_RANDOM Random method.

Definition at line 56 of file optimize.h.

```
00057 {
00058   DIRECTION_METHOD_COORDINATES = 0,
00059   DIRECTION_METHOD_RANDOM = 1,
00060 };
```

5.11.2.3 enum ErrorNorm

Enum to define the error norm.

Enumerator

ERROR_NORM_EUCLIDIAN Euclidian norm: $\sqrt{\sum_i (w_i\, x_i)^2}$.

ERROR_NORM_MAXIMUM Maximum norm: $\max_i |w_i\, x_i|$.

ERROR_NORM_P P-norm $\sqrt[p]{\sum_i |w_i\, x_i|^p}$.

ERROR_NORM_TAXICAB Taxicab norm $\sum_i |w_i\, x_i|$.

Definition at line 66 of file optimize.h.

```
00067 {
00068   ERROR_NORM_EUCLIDIAN = 0,
00070   ERROR_NORM_MAXIMUM = 1,
00072   ERROR_NORM_P = 2,
00074   ERROR_NORM_TAXICAB = 3
00076 };
```

5.11.3 Function Documentation

5.11.3.1 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 188 of file optimize.c.

```
00189 {
00190   char buffer2[64];
00191   char *buffert[MAX_NINPUTS] =
00192     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00193   xmlDoc *doc;
00194   xmlNode *node, *child;
00195   xmlChar *buffer;
00196   char *msg;
00197   int error_code;
00198   unsigned int i;
```

```
00199
00200 #if DEBUG
00201   fprintf (stderr, "input_open: start\n");
00202 #endif
00203
00204   // Resetting input data
00205   buffer = NULL;
00206   input_new ();
00207
00208   // Parsing the input file
00209 #if DEBUG
00210   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00211 #endif
00212   doc = xmlParseFile (filename);
00213   if (!doc)
00214     {
00215       msg = gettext ("Unable to parse the input file");
00216       goto exit_on_error;
00217     }
00218
00219   // Getting the root node
00220 #if DEBUG
00221   fprintf (stderr, "input_open: getting the root node\n");
00222 #endif
00223   node = xmlDocGetRootElement (doc);
00224   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00225     {
00226       msg = gettext ("Bad root XML node");
00227       goto exit_on_error;
00228     }
00229
00230   // Getting result and variables file names
00231   if (!input->result)
00232     {
00233       input->result = (char *) xmlGetProp (node, XML_RESULT);
00234       if (!input->result)
00235         input->result = (char *) xmlStrdup (result_name);
00236     }
00237   if (!input->variables)
00238     {
00239       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00240       if (!input->variables)
00241         input->variables = (char *) xmlStrdup (variables_name);
00242     }
00243
00244   // Opening simulator program name
00245   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00246   if (!input->simulator)
00247     {
00248       msg = gettext ("Bad simulator program");
00249       goto exit_on_error;
00250     }
00251
00252   // Opening evaluator program name
00253   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00254
00255   // Obtaining pseudo-random numbers generator seed
00256   input->seed
00257     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00258                                       &error_code);
00259   if (error_code)
00260     {
00261       msg = gettext ("Bad pseudo-random numbers generator seed");
00262       goto exit_on_error;
00263     }
00264
00265   // Opening algorithm
00266   buffer = xmlGetProp (node, XML_ALGORITHM);
00267   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00268     {
00269       input->algorithm = ALGORITHM_MONTE_CARLO;
00270
00271       // Obtaining simulations number
00272       input->nsimulations
00273         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00274       if (error_code)
00275         {
00276           msg = gettext ("Bad simulations number");
00277           goto exit_on_error;
00278         }
00279     }
00280   else if (!xmlStrcmp (buffer, XML_SWEEP))
00281     input->algorithm = ALGORITHM_SWEEP;
00282   else if (!xmlStrcmp (buffer, XML_GENETIC))
00283     {
00284       input->algorithm = ALGORITHM_GENETIC;
```

```
00285
00286       // Obtaining population
00287       if (xmlHasProp (node, XML_NPOPULATION))
00288         {
00289           input->nsimulations
00290             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00291           if (error_code || input->nsimulations < 3)
00292             {
00293               msg = gettext ("Invalid population number");
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           msg = gettext ("No population number");
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining generations
00304       if (xmlHasProp (node, XML_NGENERATIONS))
00305         {
00306           input->niterations
00307             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00308           if (error_code || !input->niterations)
00309             {
00310               msg = gettext ("Invalid generations number");
00311               goto exit_on_error;
00312             }
00313         }
00314       else
00315         {
00316           msg = gettext ("No generations number");
00317           goto exit_on_error;
00318         }
00319
00320       // Obtaining mutation probability
00321       if (xmlHasProp (node, XML_MUTATION))
00322         {
00323           input->mutation_ratio
00324             = xml_node_get_float (node, XML_MUTATION, &error_code);
00325           if (error_code || input->mutation_ratio < 0.
00326               || input->mutation_ratio >= 1.)
00327             {
00328               msg = gettext ("Invalid mutation probability");
00329               goto exit_on_error;
00330             }
00331         }
00332       else
00333         {
00334           msg = gettext ("No mutation probability");
00335           goto exit_on_error;
00336         }
00337
00338       // Obtaining reproduction probability
00339       if (xmlHasProp (node, XML_REPRODUCTION))
00340         {
00341           input->reproduction_ratio
00342             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00343           if (error_code || input->reproduction_ratio < 0.
00344               || input->reproduction_ratio >= 1.0)
00345             {
00346               msg = gettext ("Invalid reproduction probability");
00347               goto exit_on_error;
00348             }
00349         }
00350       else
00351         {
00352           msg = gettext ("No reproduction probability");
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining adaptation probability
00357       if (xmlHasProp (node, XML_ADAPTATION))
00358         {
00359           input->adaptation_ratio
00360             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00361           if (error_code || input->adaptation_ratio < 0.
00362               || input->adaptation_ratio >= 1.)
00363             {
00364               msg = gettext ("Invalid adaptation probability");
00365               goto exit_on_error;
00366             }
00367         }
00368       else
00369         {
00370           msg = gettext ("No adaptation probability");
00371           goto exit_on_error;
```

```
00372         }
00373
00374       // Checking survivals
00375       i = input->mutation_ratio * input->nsimulations;
00376       i += input->reproduction_ratio * input->
      nsimulations;
00377       i += input->adaptation_ratio * input->
      nsimulations;
00378       if (i > input->nsimulations - 2)
00379         {
00380           msg = gettext
00381             ("No enough survival entities to reproduce the population");
00382           goto exit_on_error;
00383         }
00384     }
00385   else
00386     {
00387       msg = gettext ("Unknown algorithm");
00388       goto exit_on_error;
00389     }
00390   xmlFree (buffer);
00391   buffer = NULL;
00392
00393   if (input->algorithm == ALGORITHM_MONTE_CARLO
00394       || input->algorithm == ALGORITHM_SWEEP)
00395     {
00396
00397       // Obtaining iterations number
00398       input->niterations
00399         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00400       if (error_code == 1)
00401         input->niterations = 1;
00402       else if (error_code)
00403         {
00404           msg = gettext ("Bad iterations number");
00405           goto exit_on_error;
00406         }
00407
00408       // Obtaining best number
00409       input->nbest
00410         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00411       if (error_code || !input->nbest)
00412         {
00413           msg = gettext ("Invalid best number");
00414           goto exit_on_error;
00415         }
00416
00417       // Obtaining tolerance
00418       input->tolerance
00419         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00420                                          &error_code);
00421       if (error_code || input->tolerance < 0.)
00422         {
00423           msg = gettext ("Invalid tolerance");
00424           goto exit_on_error;
00425         }
00426
00427       // Getting direction search method parameters
00428       if (xmlHasProp (node, XML_NSTEPS))
00429         {
00430           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00431           if (error_code || !input->nsteps)
00432             {
00433               msg = gettext ("Invalid steps number");
00434               goto exit_on_error;
00435             }
00436           buffer = xmlGetProp (node, XML_DIRECTION);
00437           if (!xmlStrcmp (buffer, XML_COORDINATES))
00438             input->direction = DIRECTION_METHOD_COORDINATES;
00439           else if (!xmlStrcmp (buffer, XML_RANDOM))
00440             {
00441               input->direction = DIRECTION_METHOD_RANDOM;
00442               input->nestimates
00443                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00444               if (error_code || !input->nestimates)
00445                 {
00446                   msg = gettext ("Invalid estimates number");
00447                   goto exit_on_error;
00448                 }
00449             }
00450           else
00451             {
00452               msg = gettext ("Unknown method to estimate the direction search");
00453               goto exit_on_error;
```

```
00454             }
00455           xmlFree (buffer);
00456           buffer = NULL;
00457           input->relaxation
00458             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00459                                                DEFAULT_RELAXATION, &error_code);
00460           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00461             {
00462               msg = gettext ("Invalid relaxation parameter");
00463               goto exit_on_error;
00464             }
00465         }
00466       else
00467         input->nsteps = 0;
00468     }
00469   // Obtaining the thresold
00470   input->thresold = xml_node_get_float_with_default (node,
      XML_THRESOLD, 0.,
00471                                                      &error_code);
00472   if (error_code)
00473     {
00474       msg = gettext ("Invalid thresold");
00475       goto exit_on_error;
00476     }
00477
00478   // Reading the experimental data
00479   for (child = node->children; child; child = child->next)
00480     {
00481       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00482         break;
00483 #if DEBUG
00484       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00485 #endif
00486       if (xmlHasProp (child, XML_NAME))
00487         buffer = xmlGetProp (child, XML_NAME);
00488       else
00489         {
00490           snprintf (buffer2, 64, "%s %u: %s",
00491                     gettext ("Experiment"),
00492                     input->nexperiments + 1, gettext ("no data file name"));
00493           msg = buffer2;
00494           goto exit_on_error;
00495         }
00496 #if DEBUG
00497       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00498 #endif
00499       input->weight = g_realloc (input->weight,
00500                                  (1 + input->nexperiments) * sizeof (double));
00501       input->weight[input->nexperiments]
00502         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00503       if (error_code)
00504         {
00505           snprintf (buffer2, 64, "%s %s: %s",
00506                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00507           msg = buffer2;
00508           goto exit_on_error;
00509         }
00510 #if DEBUG
00511       fprintf (stderr, "input_open: weight=%lg\n",
00512                input->weight[input->nexperiments]);
00513 #endif
00514       if (!input->nexperiments)
00515         input->ninputs = 0;
00516 #if DEBUG
00517       fprintf (stderr, "input_open: template[0]\n");
00518 #endif
00519       if (xmlHasProp (child, XML_TEMPLATE1))
00520         {
00521           input->template[0]
00522             = (char **) g_realloc (input->template[0],
00523                                    (1 + input->nexperiments) * sizeof (char *));
00524           buffert[0] = (char *) xmlGetProp (child, template[0]);
00525 #if DEBUG
00526           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00527                    input->nexperiments, buffert[0]);
00528 #endif
00529           if (!input->nexperiments)
00530             ++input->ninputs;
00531 #if DEBUG
00532           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00533 #endif
00534         }
00535       else
00536         {
```

```
00537           snprintf (buffer2, 64, "%s %s: %s",
00538                     gettext ("Experiment"), buffer, gettext ("no template"));
00539           msg = buffer2;
00540           goto exit_on_error;
00541         }
00542       for (i = 1; i < MAX_NINPUTS; ++i)
00543         {
00544 #if DEBUG
00545           fprintf (stderr, "input_open: template%u\n", i + 1);
00546 #endif
00547           if (xmlHasProp (child, template[i]))
00548             {
00549               if (input->nexperiments && input->ninputs <= i)
00550                 {
00551                   snprintf (buffer2, 64, "%s %s: %s",
00552                             gettext ("Experiment"),
00553                             buffer, gettext ("bad templates number"));
00554                   msg = buffer2;
00555                   while (i-- > 0)
00556                     xmlFree (buffert[i]);
00557                   goto exit_on_error;
00558                 }
00559               input->template[i] = (char **)
00560                 g_realloc (input->template[i],
00561                            (1 + input->nexperiments) * sizeof (char *));
00562               buffert[i] = (char *) xmlGetProp (child, template[i]);
00563 #if DEBUG
00564               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00565                        input->nexperiments, i + 1,
00566                        input->template[i][input->nexperiments]);
00567 #endif
00568               if (!input->nexperiments)
00569                 ++input->ninputs;
00570 #if DEBUG
00571               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00572 #endif
00573             }
00574           else if (input->nexperiments && input->ninputs > i)
00575             {
00576               snprintf (buffer2, 64, "%s %s: %s%u",
00577                         gettext ("Experiment"),
00578                         buffer, gettext ("no template"), i + 1);
00579               msg = buffer2;
00580               while (i-- > 0)
00581                 xmlFree (buffert[i]);
00582               goto exit_on_error;
00583             }
00584           else
00585             break;
00586         }
00587       input->experiment
00588         = g_realloc (input->experiment,
00589                      (1 + input->nexperiments) * sizeof (char *));
00590       input->experiment[input->nexperiments] = (char *) buffer;
00591       for (i = 0; i < input->ninputs; ++i)
00592         input->template[i][input->nexperiments] = buffert[i];
00593       ++input->nexperiments;
00594 #if DEBUG
00595       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00596 #endif
00597     }
00598   if (!input->nexperiments)
00599     {
00600       msg = gettext ("No optimization experiments");
00601       goto exit_on_error;
00602     }
00603   buffer = NULL;
00604
00605   // Reading the variables data
00606   for (; child; child = child->next)
00607     {
00608       if (xmlStrcmp (child->name, XML_VARIABLE))
00609         {
00610           snprintf (buffer2, 64, "%s %u: %s",
00611                     gettext ("Variable"),
00612                     input->nvariables + 1, gettext ("bad XML node"));
00613           msg = buffer2;
00614           goto exit_on_error;
00615         }
00616       if (xmlHasProp (child, XML_NAME))
00617         buffer = xmlGetProp (child, XML_NAME);
00618       else
00619         {
00620           snprintf (buffer2, 64, "%s %u: %s",
00621                     gettext ("Variable"),
00622                     input->nvariables + 1, gettext ("no name"));
00623           msg = buffer2;
```

```
00624           goto exit_on_error;
00625         }
00626       if (xmlHasProp (child, XML_MINIMUM))
00627         {
00628           input->rangemin = g_realloc
00629             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00630           input->rangeminabs = g_realloc
00631             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00632           input->rangemin[input->nvariables]
00633             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00634           if (error_code)
00635             {
00636               snprintf (buffer2, 64, "%s %s: %s",
00637                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00638               msg = buffer2;
00639               goto exit_on_error;
00640             }
00641           input->rangeminabs[input->nvariables]
00642             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MINIMUM,
00643                                                -G_MAXDOUBLE, &error_code);
00644           if (error_code)
00645             {
00646               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00647                         gettext ("bad absolute minimum"));
00648               msg = buffer2;
00649               goto exit_on_error;
00650             }
00651           if (input->rangemin[input->nvariables]
00652               < input->rangeminabs[input->nvariables])
00653             {
00654               snprintf (buffer2, 64, "%s %s: %s",
00655                         gettext ("Variable"),
00656                         buffer, gettext ("minimum range not allowed"));
00657               msg = buffer2;
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           snprintf (buffer2, 64, "%s %s: %s",
00664                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00665           msg = buffer2;
00666           goto exit_on_error;
00667         }
00668       if (xmlHasProp (child, XML_MAXIMUM))
00669         {
00670           input->rangemax = g_realloc
00671             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00672           input->rangemaxabs = g_realloc
00673             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00674           input->rangemax[input->nvariables]
00675             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00676           if (error_code)
00677             {
00678               snprintf (buffer2, 64, "%s %s: %s",
00679                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00680               msg = buffer2;
00681               goto exit_on_error;
00682             }
00683           input->rangemaxabs[input->nvariables]
00684             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MAXIMUM,
00685                                                G_MAXDOUBLE, &error_code);
00686           if (error_code)
00687             {
00688               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00689                         gettext ("bad absolute maximum"));
00690               msg = buffer2;
00691               goto exit_on_error;
00692             }
00693           if (input->rangemax[input->nvariables]
00694               > input->rangemaxabs[input->nvariables])
00695             {
00696               snprintf (buffer2, 64, "%s %s: %s",
00697                         gettext ("Variable"),
00698                         buffer, gettext ("maximum range not allowed"));
00699               msg = buffer2;
00700               goto exit_on_error;
00701             }
00702         }
00703       else
00704         {
00705           snprintf (buffer2, 64, "%s %s: %s",
00706                     gettext ("Variable"), buffer, gettext ("no maximum range"));
00707           msg = buffer2;
00708           goto exit_on_error;
```

```
00709         }
00710       if (input->rangemax[input->nvariables]
00711           < input->rangemin[input->nvariables])
00712         {
00713           snprintf (buffer2, 64, "%s %s: %s",
00714                     gettext ("Variable"), buffer, gettext ("bad range"));
00715           msg = buffer2;
00716           goto exit_on_error;
00717         }
00718       input->precision = g_realloc
00719         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00720       input->precision[input->nvariables]
00721         = xml_node_get_uint_with_default (child,
      XML_PRECISION,
00722                                           DEFAULT_PRECISION, &error_code);
00723       if (error_code || input->precision[input->nvariables] >=
      NPRECISIONS)
00724         {
00725           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00726                     gettext ("bad precision"));
00727           msg = buffer2;
00728           goto exit_on_error;
00729         }
00730       if (input->algorithm == ALGORITHM_SWEEP)
00731         {
00732           if (xmlHasProp (child, XML_NSWEEPS))
00733             {
00734               input->nsweeps = (unsigned int *)
00735                 g_realloc (input->nsweeps,
00736                            (1 + input->nvariables) * sizeof (unsigned int));
00737               input->nsweeps[input->nvariables]
00738                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00739               if (error_code || !input->nsweeps[input->
      nvariables])
00740                 {
00741                   snprintf (buffer2, 64, "%s %s: %s",
00742                             gettext ("Variable"),
00743                             buffer, gettext ("bad sweeps"));
00744                   msg = buffer2;
00745                   goto exit_on_error;
00746                 }
00747             }
00748           else
00749             {
00750               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00751                         gettext ("no sweeps number"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755 #if DEBUG
00756           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00757                    input->nsweeps[input->nvariables],
      input->nsimulations);
00758 #endif
00759         }
00760       if (input->algorithm == ALGORITHM_GENETIC)
00761         {
00762           // Obtaining bits representing each variable
00763           if (xmlHasProp (child, XML_NBITS))
00764             {
00765               input->nbits = (unsigned int *)
00766                 g_realloc (input->nbits,
00767                            (1 + input->nvariables) * sizeof (unsigned int));
00768               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00769               if (error_code || !i)
00770                 {
00771                   snprintf (buffer2, 64, "%s %s: %s",
00772                             gettext ("Variable"),
00773                             buffer, gettext ("invalid bits number"));
00774                   msg = buffer2;
00775                   goto exit_on_error;
00776                 }
00777               input->nbits[input->nvariables] = i;
00778             }
00779           else
00780             {
00781               snprintf (buffer2, 64, "%s %s: %s",
00782                         gettext ("Variable"),
00783                         buffer, gettext ("no bits number"));
00784               msg = buffer2;
00785               goto exit_on_error;
00786             }
00787         }
00788       else if (input->nsteps)
00789         {
00790           input->step = (double *)
00791             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
```

```
00792           input->step[input->nvariables]
00793             = xml_node_get_float (child, XML_STEP, &error_code);
00794           if (error_code || input->step[input->nvariables] < 0.)
00795             {
00796               snprintf (buffer2, 64, "%s %s: %s",
00797                         gettext ("Variable"),
00798                         buffer, gettext ("bad step size"));
00799               msg = buffer2;
00800               goto exit_on_error;
00801             }
00802         }
00803       input->label = g_realloc
00804         (input->label, (1 + input->nvariables) * sizeof (char *));
00805       input->label[input->nvariables] = (char *) buffer;
00806       ++input->nvariables;
00807     }
00808   if (!input->nvariables)
00809     {
00810       msg = gettext ("No optimization variables");
00811       goto exit_on_error;
00812     }
00813   buffer = NULL;
00814
00815   // Obtaining the error norm
00816   if (xmlHasProp (node, XML_NORM))
00817     {
00818       buffer = xmlGetProp (node, XML_NORM);
00819       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00820         input->norm = ERROR_NORM_EUCLIDIAN;
00821       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00822         input->norm = ERROR_NORM_MAXIMUM;
00823       else if (!xmlStrcmp (buffer, XML_P))
00824         {
00825           input->norm = ERROR_NORM_P;
00826           input->p = xml_node_get_float (node, XML_P, &error_code);
00827           if (!error_code)
00828             {
00829               msg = gettext ("Bad P parameter");
00830               goto exit_on_error;
00831             }
00832         }
00833       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00834         input->norm = ERROR_NORM_TAXICAB;
00835       else
00836         {
00837           msg = gettext ("Unknown error norm");
00838           goto exit_on_error;
00839         }
00840       xmlFree (buffer);
00841     }
00842   else
00843     input->norm = ERROR_NORM_EUCLIDIAN;
00844
00845   // Getting the working directory
00846   input->directory = g_path_get_dirname (filename);
00847   input->name = g_path_get_basename (filename);
00848
00849   // Closing the XML document
00850   xmlFreeDoc (doc);
00851
00852 #if DEBUG
00853   fprintf (stderr, "input_open: end\n");
00854 #endif
00855   return 1;
00856
00857 exit_on_error:
00858   xmlFree (buffer);
00859   xmlFreeDoc (doc);
00860   show_error (msg);
00861   input_free ();
00862 #if DEBUG
00863   fprintf (stderr, "input_open: end\n");
00864 #endif
00865   return 0;
00866 }
```

Here is the call graph for this function:

5.11.3.2 void optimize_best (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1236 of file optimize.c.

```
01237 {
01238   unsigned int i, j;
01239   double e;
01240 #if DEBUG
01241   fprintf (stderr, "optimize_best: start\n");
01242   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
01243            optimize->nsaveds, optimize->nbest);
01244 #endif
01245   if (optimize->nsaveds < optimize->nbest
01246       || value < optimize->error_best[optimize->nsaveds - 1])
01247     {
01248       if (optimize->nsaveds < optimize->nbest)
01249         ++optimize->nsaveds;
01250       optimize->error_best[optimize->nsaveds - 1] = value;
01251       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
01252       for (i = optimize->nsaveds; --i;)
01253         {
01254           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
01255             {
01256               j = optimize->simulation_best[i];
01257               e = optimize->error_best[i];
01258               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
01259               optimize->error_best[i] = optimize->
      error_best[i - 1];
01260               optimize->simulation_best[i - 1] = j;
01261               optimize->error_best[i - 1] = e;
01262             }
01263           else
01264             break;
01265         }
01266     }
01267 #if DEBUG
01268   fprintf (stderr, "optimize_best: end\n");
01269 #endif
01270 }
```

5.11.3.3 void optimize_best_direction (unsigned int *simulation*, double *value*)

Function to save the best simulation in a direction search method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1561 of file optimize.c.

```
01562 {
01563 #if DEBUG
01564   fprintf (stderr, "optimize_best_direction: start\n");
01565   fprintf (stderr,
01566            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
01567            simulation, value, optimize->error_best[0]);
01568 #endif
01569   if (value < optimize->error_best[0])
01570     {
01571       optimize->error_best[0] = value;
01572       optimize->simulation_best[0] = simulation;
01573 #if DEBUG
01574       fprintf (stderr,
01575                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
01576                simulation, value);
01577 #endif
01578     }
01579 #if DEBUG
01580   fprintf (stderr, "optimize_best_direction: end\n");
01581 #endif
01582 }
```

5.11.3.4 void* optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1629 of file optimize.c.

```
01630 {
01631   unsigned int i, thread;
01632   double e;
01633 #if DEBUG
01634   fprintf (stderr, "optimize_direction_thread: start\n");
01635 #endif
01636   thread = data->thread;
01637 #if DEBUG
01638   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
01639            thread,
01640            optimize->thread_direction[thread],
01641            optimize->thread_direction[thread + 1]);
01642 #endif
01643   for (i = optimize->thread_direction[thread];
01644        i < optimize->thread_direction[thread + 1]; ++i)
01645     {
01646       e = optimize_norm (i);
01647       g_mutex_lock (mutex);
01648       optimize_best_direction (i, e);
01649       optimize_save_variables (i, e);
01650       if (e < optimize->thresold)
01651         optimize->stop = 1;
01652       g_mutex_unlock (mutex);
01653       if (optimize->stop)
01654         break;
01655 #if DEBUG
01656       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
01657 #endif
01658     }
01659 #if DEBUG
01660   fprintf (stderr, "optimize_direction_thread: end\n");
01661 #endif
01662   g_thread_exit (NULL);
01663   return NULL;
01664 }
```

Here is the call graph for this function:

5.11.3.5 double optimize_estimate_direction_coordinates (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1703 of file optimize.c.

```
01705 {
01706   double x;
01707 #if DEBUG
01708   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
01709 #endif
01710   x = optimize->direction[variable];
01711   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01712     {
01713       if (estimate & 1)
01714         x += optimize->step[variable];
01715       else
01716         x -= optimize->step[variable];
01717     }
01718 #if DEBUG
01719   fprintf (stderr,
01720            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
```

```
01721            variable, x);
01722   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
01723 #endif
01724   return x;
01725 }
```

5.11.3.6 double optimize_estimate_direction_random (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1676 of file optimize.c.

```
01678 {
01679   double x;
01680 #if DEBUG
01681   fprintf (stderr, "optimize_estimate_direction_random: start\n");
01682 #endif
01683   x = optimize->direction[variable]
01684     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
01685 #if DEBUG
01686   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
01687            variable, x);
01688   fprintf (stderr, "optimize_estimate_direction_random: end\n");
01689 #endif
01690   return x;
01691 }
```

5.11.3.7 double optimize_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

Returns

objective function value.

Definition at line 1870 of file optimize.c.

```
01871 {
01872   unsigned int j;
01873   double objective;
01874   char buffer[64];
01875 #if DEBUG
01876   fprintf (stderr, "optimize_genetic_objective: start\n");
01877 #endif
01878   for (j = 0; j < optimize->nvariables; ++j)
01879     {
01880       optimize->value[entity->id * optimize->nvariables + j]
01881         = genetic_get_variable (entity, optimize->genetic_variable + j);
01882     }
01883   objective = optimize_norm (entity->id);
01884   g_mutex_lock (mutex);
01885   for (j = 0; j < optimize->nvariables; ++j)
01886     {
01887       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01888       fprintf (optimize->file_variables, buffer,
01889                genetic_get_variable (entity, optimize->genetic_variable + j));
01890     }
01891   fprintf (optimize->file_variables, "%.14le\n", objective);
01892   g_mutex_unlock (mutex);
01893 #if DEBUG
01894   fprintf (stderr, "optimize_genetic_objective: end\n");
01895 #endif
01896   return objective;
01897 }
```

5.11.3.8 void optimize_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 880 of file optimize.c.

```
00881 {
00882   unsigned int i;
00883   char buffer[32], value[32], *buffer2, *buffer3, *content;
00884   FILE *file;
00885   gsize length;
00886   GRegex *regex;
00887 
00888 #if DEBUG
00889   fprintf (stderr, "optimize_input: start\n");
00890 #endif
00891 
00892   // Checking the file
00893   if (!template)
00894     goto optimize_input_end;
00895 
00896   // Opening template
00897   content = g_mapped_file_get_contents (template);
00898   length = g_mapped_file_get_length (template);
00899 #if DEBUG
00900   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00901 #endif
00902   file = g_fopen (input, "w");
00903 
00904   // Parsing template
00905   for (i = 0; i < optimize->nvariables; ++i)
00906     {
00907 #if DEBUG
00908       fprintf (stderr, "optimize_input: variable=%u\n", i);
00909 #endif
00910       snprintf (buffer, 32, "@variable%u@", i + 1);
00911       regex = g_regex_new (buffer, 0, 0, NULL);
00912       if (i == 0)
00913         {
00914           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00915                                              optimize->label[i], 0, NULL);
00916 #if DEBUG
00917           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00918 #endif
00919         }
00920       else
00921         {
00922           length = strlen (buffer3);
00923           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00924                                              optimize->label[i], 0, NULL);
00925           g_free (buffer3);
00926         }
00927       g_regex_unref (regex);
00928       length = strlen (buffer2);
00929       snprintf (buffer, 32, "@value%u@", i + 1);
00930       regex = g_regex_new (buffer, 0, 0, NULL);
00931       snprintf (value, 32, format[optimize->precision[i]],
00932                 optimize->value[simulation * optimize->
      nvariables + i]);
00933 
00934 #if DEBUG
00935       fprintf (stderr, "optimize_input: value=%s\n", value);
00936 #endif
00937       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00938                                          0, NULL);
00939       g_free (buffer2);
00940       g_regex_unref (regex);
00941     }
00942 
00943   // Saving input file
00944   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00945   g_free (buffer3);
00946   fclose (file);
00947 
00948 optimize_input_end:
00949 #if DEBUG
00950   fprintf (stderr, "optimize_input: end\n");
```

```
00951 #endif
00952   return;
00953 }
```

5.11.3.9 void optimize_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 optimization results.

Parameters

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1359 of file optimize.c.

```
01361 {
01362   unsigned int i, j, k, s[optimize->nbest];
01363   double e[optimize->nbest];
01364 #if DEBUG
01365   fprintf (stderr, "optimize_merge: start\n");
01366 #endif
01367   i = j = k = 0;
01368   do
01369     {
01370       if (i == optimize->nsaveds)
01371         {
01372           s[k] = simulation_best[j];
01373           e[k] = error_best[j];
01374           ++j;
01375           ++k;
01376           if (j == nsaveds)
01377             break;
01378         }
01379       else if (j == nsaveds)
01380         {
01381           s[k] = optimize->simulation_best[i];
01382           e[k] = optimize->error_best[i];
01383           ++i;
01384           ++k;
01385           if (i == optimize->nsaveds)
01386             break;
01387         }
01388       else if (optimize->error_best[i] > error_best[j])
01389         {
01390           s[k] = simulation_best[j];
01391           e[k] = error_best[j];
01392           ++j;
01393           ++k;
01394         }
01395       else
01396         {
01397           s[k] = optimize->simulation_best[i];
01398           e[k] = optimize->error_best[i];
01399           ++i;
01400           ++k;
01401         }
01402     }
01403   while (k < optimize->nbest);
01404   optimize->nsaveds = k;
01405   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
01406   memcpy (optimize->error_best, e, k * sizeof (double));
01407 #if DEBUG
01408   fprintf (stderr, "optimize_merge: end\n");
01409 #endif
01410 }
```

5.11.3.10 double optimize_norm_euclidian (unsigned int *simulation*)

Function to calculate the Euclidian error norm.

Parameters

simulation	simulation number.

Returns

Euclidian error norm.

Definition at line 1069 of file optimize.c.

```
01070 {
01071   double e, ei;
01072   unsigned int i;
01073 #if DEBUG
01074   fprintf (stderr, "optimize_norm_euclidian: start\n");
01075 #endif
01076   e = 0.;
01077   for (i = 0; i < optimize->nexperiments; ++i)
01078     {
01079       ei = optimize_parse (simulation, i);
01080       e += ei * ei;
01081     }
01082   e = sqrt (e);
01083 #if DEBUG
01084   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
01085   fprintf (stderr, "optimize_norm_euclidian: end\n");
01086 #endif
01087   return e;
01088 }
```

Here is the call graph for this function:

5.11.3.11 double optimize_norm_maximum (unsigned int *simulation*)

Function to calculate the maximum error norm.

Parameters

simulation	simulation number.

Returns

Maximum error norm.

Definition at line 1098 of file optimize.c.

```
01099 {
01100   double e, ei;
01101   unsigned int i;
01102 #if DEBUG
01103   fprintf (stderr, "optimize_norm_maximum: start\n");
01104 #endif
01105   e = 0.;
01106   for (i = 0; i < optimize->nexperiments; ++i)
01107     {
01108       ei = fabs (optimize_parse (simulation, i));
01109       e = fmax (e, ei);
01110     }
01111 #if DEBUG
01112   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
01113   fprintf (stderr, "optimize_norm_maximum: end\n");
01114 #endif
01115   return e;
01116 }
```

Here is the call graph for this function:

5.11.3.12 double optimize_norm_p (unsigned int *simulation*)

Function to calculate the P error norm.

Parameters

simulation	simulation number.

Returns

P error norm.

Definition at line 1126 of file optimize.c.

```
01127 {
01128   double e, ei;
01129   unsigned int i;
01130 #if DEBUG
01131   fprintf (stderr, "optimize_norm_p: start\n");
01132 #endif
01133   e = 0.;
01134   for (i = 0; i < optimize->nexperiments; ++i)
01135     {
01136       ei = fabs (optimize_parse (simulation, i));
01137       e += pow (ei, optimize->p);
01138     }
01139   e = pow (e, 1. / optimize->p);
01140 #if DEBUG
01141   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
01142   fprintf (stderr, "optimize_norm_p: end\n");
01143 #endif
01144   return e;
01145 }
```

Here is the call graph for this function:

5.11.3.13 double optimize_norm_taxicab (unsigned int *simulation*)

Function to calculate the taxicab error norm.

Parameters

simulation	simulation number.

Returns

Taxicab error norm.

Definition at line 1155 of file optimize.c.

```
01156 {
01157   double e;
01158   unsigned int i;
01159 #if DEBUG
01160   fprintf (stderr, "optimize_norm_taxicab: start\n");
01161 #endif
01162   e = 0.;
01163   for (i = 0; i < optimize->nexperiments; ++i)
01164     e += fabs (optimize_parse (simulation, i));
01165 #if DEBUG
01166   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
01167   fprintf (stderr, "optimize_norm_taxicab: end\n");
01168 #endif
01169   return e;
01170 }
```

Here is the call graph for this function:

5.11.3.14 double optimize_parse (unsigned int *simulation*, unsigned int *experiment*)

Function to parse input files, simulating and calculating the \ objective function.

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 966 of file optimize.c.

```
00967 {
00968   unsigned int i;
00969   double e;
00970   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00971     *buffer3, *buffer4;
00972   FILE *file_result;
00973 
00974 #if DEBUG
00975   fprintf (stderr, "optimize_parse: start\n");
00976   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00977            experiment);
00978 #endif
00979 
00980   // Opening input files
00981   for (i = 0; i < optimize->ninputs; ++i)
00982     {
00983       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00984 #if DEBUG
00985       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00986 #endif
00987       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00988     }
00989   for (; i < MAX_NINPUTS; ++i)
00990     strcpy (&input[i][0], "");
00991 #if DEBUG
00992   fprintf (stderr, "optimize_parse: parsing end\n");
00993 #endif
00994 
00995   // Performing the simulation
00996   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00997   buffer2 = g_path_get_dirname (optimize->simulator);
00998   buffer3 = g_path_get_basename (optimize->simulator);
00999   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01000   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01001             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01002             input[6], input[7], output);
01003   g_free (buffer4);
01004   g_free (buffer3);
01005   g_free (buffer2);
01006 #if DEBUG
01007   fprintf (stderr, "optimize_parse: %s\n", buffer);
01008 #endif
01009   system (buffer);
01010 
01011   // Checking the objective value function
01012   if (optimize->evaluator)
01013     {
01014       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01015       buffer2 = g_path_get_dirname (optimize->evaluator);
01016       buffer3 = g_path_get_basename (optimize->evaluator);
01017       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01018       snprintf (buffer, 512, "\"%s\" %s %s %s",
01019                 buffer4, output, optimize->experiment[experiment], result);
01020       g_free (buffer4);
01021       g_free (buffer3);
01022       g_free (buffer2);
01023 #if DEBUG
01024       fprintf (stderr, "optimize_parse: %s\n", buffer);
01025 #endif
01026       system (buffer);
01027       file_result = g_fopen (result, "r");
01028       e = atof (fgets (buffer, 512, file_result));
01029       fclose (file_result);
01030     }
01031   else
01032     {
01033       strcpy (result, "");
01034       file_result = g_fopen (output, "r");
01035       e = atof (fgets (buffer, 512, file_result));
01036       fclose (file_result);
```

```
01037     }
01038
01039   // Removing files
01040 #if !DEBUG
01041   for (i = 0; i < optimize->ninputs; ++i)
01042     {
01043       if (optimize->file[i][0])
01044         {
01045           snprintf (buffer, 512, RM " %s", &input[i][0]);
01046           system (buffer);
01047         }
01048     }
01049   snprintf (buffer, 512, RM " %s %s", output, result);
01050   system (buffer);
01051 #endif
01052
01053 #if DEBUG
01054   fprintf (stderr, "optimize_parse: end\n");
01055 #endif
01056
01057   // Returning the objective function
01058   return e * optimize->weight[experiment];
01059 }
```

Here is the call graph for this function:

5.11.3.15 void optimize_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1208 of file optimize.c.

```
01209 {
01210   unsigned int i;
01211   char buffer[64];
01212 #if DEBUG
01213   fprintf (stderr, "optimize_save_variables: start\n");
01214 #endif
01215   for (i = 0; i < optimize->nvariables; ++i)
01216     {
01217       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
01218       fprintf (optimize->file_variables, buffer,
01219                optimize->value[simulation * optimize->
      nvariables + i]);
01220     }
01221   fprintf (optimize->file_variables, "%.14le\n", error);
01222 #if DEBUG
01223   fprintf (stderr, "optimize_save_variables: end\n");
01224 #endif
01225 }
```

5.11.3.16 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 1734 of file optimize.c.

```
01735 {
01736   GThread *thread[nthreads_direction];
01737   ParallelData data[nthreads_direction];
01738   unsigned int i, j, k, b;
01739 #if DEBUG
01740   fprintf (stderr, "optimize_step_direction: start\n");
01741 #endif
01742   for (i = 0; i < optimize->nestimates; ++i)
```

```
01743     {
01744       k = (simulation + i) * optimize->nvariables;
01745       b = optimize->simulation_best[0] * optimize->
      nvariables;
01746 #if DEBUG
01747       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
01748                simulation + i, optimize->simulation_best[0]);
01749 #endif
01750       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
01751         {
01752 #if DEBUG
01753           fprintf (stderr,
01754                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
01755                    i, j, optimize->value[b]);
01756 #endif
01757           optimize->value[k]
01758             = optimize->value[b] + optimize_estimate_direction (j,
      i);
01759           optimize->value[k] = fmin (fmax (optimize->value[k],
01760                                            optimize->rangeminabs[j]),
01761                                      optimize->rangemaxabs[j]);
01762 #if DEBUG
01763           fprintf (stderr,
01764                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
01765                    i, j, optimize->value[k]);
01766 #endif
01767         }
01768     }
01769   if (nthreads_direction == 1)
01770     optimize_direction_sequential (simulation);
01771   else
01772     {
01773       for (i = 0; i <= nthreads_direction; ++i)
01774         {
01775           optimize->thread_direction[i]
01776             = simulation + optimize->nstart_direction
01777             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01778             / nthreads_direction;
01779 #if DEBUG
01780           fprintf (stderr,
01781                    "optimize_step_direction: i=%u thread_direction=%u\n",
01782                    i, optimize->thread_direction[i]);
01783 #endif
01784         }
01785       for (i = 0; i < nthreads_direction; ++i)
01786         {
01787           data[i].thread = i;
01788           thread[i] = g_thread_new
01789             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01790         }
01791       for (i = 0; i < nthreads_direction; ++i)
01792         g_thread_join (thread[i]);
01793     }
01794 #if DEBUG
01795   fprintf (stderr, "optimize_step_direction: end\n");
01796 #endif
01797 }
```

Here is the call graph for this function:

5.11.3.17 void* optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1313 of file optimize.c.

```
01314 {
01315   unsigned int i, thread;
01316   double e;
```

```
01317 #if DEBUG
01318   fprintf (stderr, "optimize_thread: start\n");
01319 #endif
01320   thread = data->thread;
01321 #if DEBUG
01322   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
01323            optimize->thread[thread], optimize->thread[thread + 1]);
01324 #endif
01325   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
01326     {
01327       e = optimize_norm (i);
01328       g_mutex_lock (mutex);
01329       optimize_best (i, e);
01330       optimize_save_variables (i, e);
01331       if (e < optimize->thresold)
01332         optimize->stop = 1;
01333       g_mutex_unlock (mutex);
01334       if (optimize->stop)
01335         break;
01336 #if DEBUG
01337       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
01338 #endif
01339     }
01340 #if DEBUG
01341   fprintf (stderr, "optimize_thread: end\n");
01342 #endif
01343   g_thread_exit (NULL);
01344   return NULL;
01345 }
```

Here is the call graph for this function:

5.12 optimize.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef OPTIMIZE__H
00039 #define OPTIMIZE__H 1
00040
00045 enum Algorithm
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
00051
00056 enum DirectionMethod
00057 {
00058   DIRECTION_METHOD_COORDINATES = 0,
00059   DIRECTION_METHOD_RANDOM = 1,
00060 };
00061
00066 enum ErrorNorm
```

```
00067 {
00068   ERROR_NORM_EUCLIDIAN = 0,
00070   ERROR_NORM_MAXIMUM = 1,
00072   ERROR_NORM_P = 2,
00074   ERROR_NORM_TAXICAB = 3
00076 };
00077
00082 typedef struct
00083 {
00084   char **template[MAX_NINPUTS];
00085   char **experiment;
00086   char **label;
00087   char *result;
00088   char *variables;
00089   char *simulator;
00090   char *evaluator;
00092   char *directory;
00093   char *name;
00094   double *rangemin;
00095   double *rangemax;
00096   double *rangeminabs;
00097   double *rangemaxabs;
00098   double *weight;
00099   double *step;
00101   unsigned int *precision;
00102   unsigned int *nsweeps;
00103   unsigned int *nbits;
00105   double tolerance;
00106   double mutation_ratio;
00107   double reproduction_ratio;
00108   double adaptation_ratio;
00109   double relaxation;
00110   double p;
00111   double thresold;
00112   unsigned long int seed;
00114   unsigned int nvariables;
00115   unsigned int nexperiments;
00116   unsigned int ninputs;
00117   unsigned int nsimulations;
00118   unsigned int algorithm;
00119   unsigned int nsteps;
00121   unsigned int direction;
00122   unsigned int nestimates;
00124   unsigned int niterations;
00125   unsigned int nbest;
00126   unsigned int norm;
00127 } Input;
00128
00133 typedef struct
00134 {
00135   GMappedFile **file[MAX_NINPUTS];
00136   char **template[MAX_NINPUTS];
00137   char **experiment;
00138   char **label;
00139   gsl_rng *rng;
00140   GeneticVariable *genetic_variable;
00142   FILE *file_result;
00143   FILE *file_variables;
00144   char *result;
00145   char *variables;
00146   char *simulator;
00147   char *evaluator;
00149   double *value;
00150   double *rangemin;
00151   double *rangemax;
00152   double *rangeminabs;
00153   double *rangemaxabs;
00154   double *error_best;
00155   double *weight;
00156   double *step;
00158   double *direction;
00159   double *value_old;
00161   double *error_old;
00163   unsigned int *precision;
00164   unsigned int *nsweeps;
00165   unsigned int *thread;
00167   unsigned int *thread_direction;
00170   unsigned int *simulation_best;
00171   double tolerance;
00172   double mutation_ratio;
00173   double reproduction_ratio;
00174   double adaptation_ratio;
00175   double relaxation;
00176   double calculation_time;
00177   double p;
00178   double thresold;
00179   unsigned long int seed;
```

```
00181   unsigned int nvariables;
00182   unsigned int nexperiments;
00183   unsigned int ninputs;
00184   unsigned int nsimulations;
00185   unsigned int nsteps;
00187   unsigned int nestimates;
00189   unsigned int algorithm;
00190   unsigned int nstart;
00191   unsigned int nend;
00192   unsigned int nstart_direction;
00194   unsigned int nend_direction;
00196   unsigned int niterations;
00197   unsigned int nbest;
00198   unsigned int nsaveds;
00199   unsigned int stop;
00200 #if HAVE_MPI
00201   int mpi_rank;
00202 #endif
00203 } Optimize;
00204
00209 typedef struct
00210 {
00211   unsigned int thread;
00212 } ParallelData;
00213
00214 // Global variables
00215 extern int ntasks;
00216 extern unsigned int nthreads;
00217 extern unsigned int nthreads_direction;
00218 extern GMutex mutex[1];
00219 extern void (*optimize_algorithm) ();
00220 extern double (*optimize_estimate_direction) (unsigned int variable,
00221                                               unsigned int estimate);
00222 extern double (*optimize_norm) (unsigned int simulation);
00223 extern Input input[1];
00224 extern Optimize optimize[1];
00225 extern const xmlChar *result_name;
00226 extern const xmlChar *variables_name;
00227 extern const xmlChar *template[MAX_NINPUTS];
00228 extern const char *format[NPRECISIONS];
00229 extern const double precision[NPRECISIONS];
00230
00231 // Public functions
00232 void input_new ();
00233 void input_free ();
00234 int input_open (char *filename);
00235 void optimize_input (unsigned int simulation, char *input,
00236                      GMappedFile * template);
00237 double optimize_parse (unsigned int simulation, unsigned int experiment);
00238 double optimize_norm_euclidian (unsigned int simulation);
00239 double optimize_norm_maximum (unsigned int simulation);
00240 double optimize_norm_p (unsigned int simulation);
00241 double optimize_norm_taxicab (unsigned int simulation);
00242 void optimize_print ();
00243 void optimize_save_variables (unsigned int simulation, double error);
00244 void optimize_best (unsigned int simulation, double value);
00245 void optimize_sequential ();
00246 void *optimize_thread (ParallelData * data);
00247 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00248                      double *error_best);
00249 #if HAVE_MPI
00250 void optimize_synchronise ();
00251 #endif
00252 void optimize_sweep ();
00253 void optimize_MonteCarlo ();
00254 void optimize_best_direction (unsigned int simulation, double value);
00255 void optimize_direction_sequential ();
00256 void *optimize_direction_thread (ParallelData * data);
00257 double optimize_estimate_direction_random (unsigned int variable,
00258                                            unsigned int estimate);
00259 double optimize_estimate_direction_coordinates (unsigned int
      variable,
00260                                                 unsigned int estimate);
00261 void optimize_step_direction (unsigned int simulation);
00262 void optimize_direction ();
00263 double optimize_genetic_objective (Entity * entity);
00264 void optimize_genetic ();
00265 void optimize_save_old ();
00266 void optimize_merge_old ();
00267 void optimize_refine ();
00268 void optimize_step ();
00269 void optimize_iterate ();
00270 void optimize_free ();
00271 void optimize_open ();
00272
00273 #endif
```

5.13 utils.c File Reference

Source file to define some useful functions.

```
#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <glib.h>
#include <libintl.h>
#include <gtk/gtk.h>
#include "utils.h"
```

Include dependency graph for utils.c:

Functions

- void show_message (char *title, char *msg, int type)

  Function to show a dialog with a message.
- void show_error (char *msg)

  Function to show a dialog with an error message.
- int xml_node_get_int (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an integer number of a XML node property.
- unsigned int xml_node_get_uint (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an unsigned integer number of a XML node property.
- unsigned int xml_node_get_uint_with_default (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)

  Function to get an unsigned integer number of a XML node property with a default value.
- double xml_node_get_float (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get a floating point number of a XML node property.
- double xml_node_get_float_with_default (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)

  Function to get a floating point number of a XML node property with a default value.
- void xml_node_set_int (xmlNode *node, const xmlChar *prop, int value)

  Function to set an integer number in a XML node property.
- void xml_node_set_uint (xmlNode *node, const xmlChar *prop, unsigned int value)

  Function to set an unsigned integer number in a XML node property.
- void xml_node_set_float (xmlNode *node, const xmlChar *prop, double value)

  Function to set a floating point number in a XML node property.
- int cores_number ()

  Function to obtain the cores number.
- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.

Variables

- GtkWindow * main_window

  Main GtkWindow.

5.13.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file utils.c.

5.13.2 Function Documentation

5.13.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 318 of file utils.c.

```
00319 {
00320 #ifdef G_OS_WIN32
00321   SYSTEM_INFO sysinfo;
00322   GetSystemInfo (&sysinfo);
00323   return sysinfo.dwNumberOfProcessors;
00324 #else
00325   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00326 #endif
00327 }
```

5.13.2.2 unsigned int gtk_array_get_active (GtkRadioButton * *array[ ]*, unsigned int *n*)

Function to get the active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 342 of file utils.c.

```
00343 {
00344   unsigned int i;
00345   for (i = 0; i < n; ++i)
00346     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00347       break;
00348   return i;
00349 }
```

5.13.2.3 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 98 of file utils.c.

```
00099 {
00100   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00101 }
```

Here is the call graph for this function:

5.13.2.4 void show_message (char * *title*, char * *msg*, int *type*)

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 68 of file utils.c.

```
00069 {
00070 #if HAVE_GTK
00071   GtkMessageDialog *dlg;
00072
00073   // Creating the dialog
00074   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00075     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00076
00077   // Setting the dialog title
00078   gtk_window_set_title (GTK_WINDOW (dlg), title);
00079
00080   // Showing the dialog and waiting response
00081   gtk_dialog_run (GTK_DIALOG (dlg));
00082
00083   // Closing and freeing memory
00084   gtk_widget_destroy (GTK_WIDGET (dlg));
00085
00086 #else
00087   printf ("%s: %s\n", title, msg);
00088 #endif
00089 }
```

5.13.2.5 double xml_node_get_float (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get a floating point number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 208 of file utils.c.

```
00209 {
00210   double x = 0.;
00211   xmlChar *buffer;
00212   buffer = xmlGetProp (node, prop);
00213   if (!buffer)
```

```
00214     *error_code = 1;
00215   else
00216     {
00217       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00218         *error_code = 2;
00219       else
00220         *error_code = 0;
00221       xmlFree (buffer);
00222     }
00223   return x;
00224 }
```

5.13.2.6 double xml_node_get_float_with_default (xmlNode * *node*, const xmlChar * *prop*, double *default_value*, int * *error_code*)

Function to get a floating point number of a XML node property with a default value.

Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Returns

Floating point number value.

Definition at line 242 of file utils.c.

```
00244 {
00245   double x;
00246   if (xmlHasProp (node, prop))
00247     x = xml_node_get_float (node, prop, error_code);
00248   else
00249     {
00250       x = default_value;
00251       *error_code = 0;
00252     }
00253   return x;
00254 }
```

Here is the call graph for this function:

5.13.2.7 int xml_node_get_int (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 116 of file utils.c.

```
00117 {
00118   int i = 0;
00119   xmlChar *buffer;
00120   buffer = xmlGetProp (node, prop);
00121   if (!buffer)
```

```
00122     *error_code = 1;
00123   else
00124     {
00125       if (sscanf ((char *) buffer, "%d", &i) != 1)
00126         *error_code = 2;
00127       else
00128         *error_code = 0;
00129       xmlFree (buffer);
00130     }
00131   return i;
00132 }
```

5.13.2.8 int xml_node_get_uint (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an unsigned integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 147 of file utils.c.

```
00148 {
00149   unsigned int i = 0;
00150   xmlChar *buffer;
00151   buffer = xmlGetProp (node, prop);
00152   if (!buffer)
00153     *error_code = 1;
00154   else
00155     {
00156       if (sscanf ((char *) buffer, "%u", &i) != 1)
00157         *error_code = 2;
00158       else
00159         *error_code = 0;
00160       xmlFree (buffer);
00161     }
00162   return i;
00163 }
```

5.13.2.9 int xml_node_get_uint_with_default (xmlNode * *node*, const xmlChar * *prop*, unsigned int *default_value*, int * *error_code*)

Function to get an unsigned integer number of a XML node property with a default value.

Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 181 of file utils.c.

```
00183 {
00184   unsigned int i;
00185   if (xmlHasProp (node, prop))
```

```
00186     i = xml_node_get_uint (node, prop, error_code);
00187   else
00188     {
00189       i = default_value;
00190       *error_code = 0;
00191     }
00192   return i;
00193 }
```

Here is the call graph for this function:

5.13.2.10 void xml_node_set_float (xmlNode * *node*, const xmlChar * *prop*, double *value*)

Function to set a floating point number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 305 of file utils.c.

```
00306 {
00307   xmlChar buffer[64];
00308   snprintf ((char *) buffer, 64, "%.14lg", value);
00309   xmlSetProp (node, prop, buffer);
00310 }
```

5.13.2.11 void xml_node_set_int (xmlNode * *node*, const xmlChar * *prop*, int *value*)

Function to set an integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 267 of file utils.c.

```
00268 {
00269   xmlChar buffer[64];
00270   snprintf ((char *) buffer, 64, "%d", value);
00271   xmlSetProp (node, prop, buffer);
00272 }
```

5.13.2.12 void xml_node_set_uint (xmlNode * *node*, const xmlChar * *prop*, unsigned int *value*)

Function to set an unsigned integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Unsigned integer number value.

Definition at line 286 of file utils.c.

```
00287 {
00288   xmlChar buffer[64];
00289   snprintf ((char *) buffer, 64, "%u", value);
00290   xmlSetProp (node, prop, buffer);
00291 }
```

5.14 utils.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <unistd.h>
00042 #include <libxml/parser.h>
00043 #include <glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #include <libintl.h>
00048 #if HAVE_GTK
00049 #include <gtk/gtk.h>
00050 #endif
00051 #include "utils.h"
00052
00053 #if HAVE_GTK
00054 GtkWindow *main_window;
00055 #endif
00056
00067 void
00068 show_message (char *title, char *msg, int type)
00069 {
00070 #if HAVE_GTK
00071   GtkMessageDialog *dlg;
00072
00073   // Creating the dialog
00074   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00075     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00076
00077   // Setting the dialog title
00078   gtk_window_set_title (GTK_WINDOW (dlg), title);
00079
00080   // Showing the dialog and waiting response
00081   gtk_dialog_run (GTK_DIALOG (dlg));
00082
00083   // Closing and freeing memory
00084   gtk_widget_destroy (GTK_WIDGET (dlg));
00085
00086 #else
00087   printf ("%s: %s\n", title, msg);
00088 #endif
00089 }
00090
00097 void
00098 show_error (char *msg)
00099 {
00100   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00101 }
00102
00115 int
00116 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00117 {
00118   int i = 0;
```

```
00119   xmlChar *buffer;
00120   buffer = xmlGetProp (node, prop);
00121   if (!buffer)
00122     *error_code = 1;
00123   else
00124     {
00125       if (sscanf ((char *) buffer, "%d", &i) != 1)
00126         *error_code = 2;
00127       else
00128         *error_code = 0;
00129       xmlFree (buffer);
00130     }
00131   return i;
00132 }
00133
00146 unsigned int
00147 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00148 {
00149   unsigned int i = 0;
00150   xmlChar *buffer;
00151   buffer = xmlGetProp (node, prop);
00152   if (!buffer)
00153     *error_code = 1;
00154   else
00155     {
00156       if (sscanf ((char *) buffer, "%u", &i) != 1)
00157         *error_code = 2;
00158       else
00159         *error_code = 0;
00160       xmlFree (buffer);
00161     }
00162   return i;
00163 }
00164
00180 unsigned int
00181 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00182                                 unsigned int default_value, int *error_code)
00183 {
00184   unsigned int i;
00185   if (xmlHasProp (node, prop))
00186     i = xml_node_get_uint (node, prop, error_code);
00187   else
00188     {
00189       i = default_value;
00190       *error_code = 0;
00191     }
00192   return i;
00193 }
00194
00207 double
00208 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00209 {
00210   double x = 0.;
00211   xmlChar *buffer;
00212   buffer = xmlGetProp (node, prop);
00213   if (!buffer)
00214     *error_code = 1;
00215   else
00216     {
00217       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00218         *error_code = 2;
00219       else
00220         *error_code = 0;
00221       xmlFree (buffer);
00222     }
00223   return x;
00224 }
00225
00241 double
00242 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00243                                  double default_value, int *error_code)
00244 {
00245   double x;
00246   if (xmlHasProp (node, prop))
00247     x = xml_node_get_float (node, prop, error_code);
00248   else
00249     {
00250       x = default_value;
00251       *error_code = 0;
00252     }
00253   return x;
00254 }
00255
00266 void
00267 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00268 {
00269   xmlChar buffer[64];
```

```
00270   snprintf ((char *) buffer, 64, "%d", value);
00271   xmlSetProp (node, prop, buffer);
00272 }
00273
00285 void
00286 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00287 {
00288   xmlChar buffer[64];
00289   snprintf ((char *) buffer, 64, "%u", value);
00290   xmlSetProp (node, prop, buffer);
00291 }
00292
00304 void
00305 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00306 {
00307   xmlChar buffer[64];
00308   snprintf ((char *) buffer, 64, "%.14lg", value);
00309   xmlSetProp (node, prop, buffer);
00310 }
00311
00317 int
00318 cores_number ()
00319 {
00320 #ifdef G_OS_WIN32
00321   SYSTEM_INFO sysinfo;
00322   GetSystemInfo (&sysinfo);
00323   return sysinfo.dwNumberOfProcessors;
00324 #else
00325   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00326 #endif
00327 }
00328
00329 #if HAVE_GTK
00330
00341 unsigned int
00342 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00343 {
00344   unsigned int i;
00345   for (i = 0; i < n; ++i)
00346     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00347       break;
00348   return i;
00349 }
00350
00351 #endif
```

5.15 utils.h File Reference

Header file to define some useful functions.

This graph shows which files directly or indirectly include this file:

Macros

- #define ERROR_TYPE GTK_MESSAGE_ERROR

  Macro to define the error message type.
- #define INFO_TYPE GTK_MESSAGE_INFO

  Macro to define the information message type.

Functions

- void show_message (char *title, char *msg, int type)

  Function to show a dialog with a message.
- void show_error (char *msg)

  Function to show a dialog with an error message.
- int xml_node_get_int (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an integer number of a XML node property.
- unsigned int xml_node_get_uint (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an unsigned integer number of a XML node property.

- unsigned int xml_node_get_uint_with_default (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)

  Function to get an unsigned integer number of a XML node property with a default value.
- double xml_node_get_float (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get a floating point number of a XML node property.
- double xml_node_get_float_with_default (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)

  Function to get a floating point number of a XML node property with a default value.
- void xml_node_set_int (xmlNode *node, const xmlChar *prop, int value)

  Function to set an integer number in a XML node property.
- void xml_node_set_uint (xmlNode *node, const xmlChar *prop, unsigned int value)

  Function to set an unsigned integer number in a XML node property.
- void xml_node_set_float (xmlNode *node, const xmlChar *prop, double value)

  Function to set a floating point number in a XML node property.
- int cores_number ()

  Function to obtain the cores number.
- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.

Variables

- GtkWindow * main_window

  Main GtkWindow.

5.15.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Definition in file utils.h.

5.15.2 Function Documentation

5.15.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 318 of file utils.c.

```
00319 {
00320 #ifdef G_OS_WIN32
00321   SYSTEM_INFO sysinfo;
00322   GetSystemInfo (&sysinfo);
00323   return sysinfo.dwNumberOfProcessors;
00324 #else
00325   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00326 #endif
00327 }
```

5.15.2.2 unsigned int gtk_array_get_active (GtkRadioButton * *array[ ]*, unsigned int *n*)

Function to get the active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 342 of file utils.c.

```
00343 {
00344   unsigned int i;
00345   for (i = 0; i < n; ++i)
00346     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00347       break;
00348   return i;
00349 }
```

5.15.2.3 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 98 of file utils.c.

```
00099 {
00100   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00101 }
```

Here is the call graph for this function:

5.15.2.4 void show_message (char * *title*, char * *msg*, int *type*)

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 68 of file utils.c.

```
00069 {
00070 #if HAVE_GTK
00071   GtkMessageDialog *dlg;
00072
00073   // Creating the dialog
00074   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00075     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00076
00077   // Setting the dialog title
00078   gtk_window_set_title (GTK_WINDOW (dlg), title);
00079
00080   // Showing the dialog and waiting response
00081   gtk_dialog_run (GTK_DIALOG (dlg));
00082
00083   // Closing and freeing memory
```

```
00084   gtk_widget_destroy (GTK_WIDGET (dlg));
00085
00086 #else
00087   printf ("%s: %s\n", title, msg);
00088 #endif
00089 }
```

5.15.2.5 double xml_node_get_float (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get a floating point number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 208 of file utils.c.

```
00209 {
00210   double x = 0.;
00211   xmlChar *buffer;
00212   buffer = xmlGetProp (node, prop);
00213   if (!buffer)
00214     *error_code = 1;
00215   else
00216     {
00217       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00218         *error_code = 2;
00219       else
00220         *error_code = 0;
00221       xmlFree (buffer);
00222     }
00223   return x;
00224 }
```

5.15.2.6 double xml_node_get_float_with_default (xmlNode * *node*, const xmlChar * *prop*, double *default_value*, int * *error_code*)

Function to get a floating point number of a XML node property with a default value.

Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Returns

Floating point number value.

Definition at line 242 of file utils.c.

```
00244 {
00245   double x;
00246   if (xmlHasProp (node, prop))
00247     x = xml_node_get_float (node, prop, error_code);
00248   else
00249     {
00250       x = default_value;
00251       *error_code = 0;
00252     }
00253   return x;
00254 }
```

Here is the call graph for this function:

5.15.2.7 int xml_node_get_int (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 116 of file utils.c.

```
00117 {
00118   int i = 0;
00119   xmlChar *buffer;
00120   buffer = xmlGetProp (node, prop);
00121   if (!buffer)
00122     *error_code = 1;
00123   else
00124     {
00125       if (sscanf ((char *) buffer, "%d", &i) != 1)
00126         *error_code = 2;
00127       else
00128         *error_code = 0;
00129       xmlFree (buffer);
00130     }
00131   return i;
00132 }
```

5.15.2.8 unsigned int xml_node_get_uint (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an unsigned integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 147 of file utils.c.

```
00148 {
00149   unsigned int i = 0;
00150   xmlChar *buffer;
00151   buffer = xmlGetProp (node, prop);
00152   if (!buffer)
00153     *error_code = 1;
00154   else
00155     {
00156       if (sscanf ((char *) buffer, "%u", &i) != 1)
00157         *error_code = 2;
00158       else
00159         *error_code = 0;
00160       xmlFree (buffer);
00161     }
00162   return i;
00163 }
```

5.15.2.9 unsigned int xml_node_get_uint_with_default (xmlNode * *node,* const xmlChar * *prop,* unsigned int *default_value,* int * *error_code*)

Function to get an unsigned integer number of a XML node property with a default value.

Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 181 of file utils.c.

```
00183 {
00184   unsigned int i;
00185   if (xmlHasProp (node, prop))
00186     i = xml_node_get_uint (node, prop, error_code);
00187   else
00188     {
00189       i = default_value;
00190       *error_code = 0;
00191     }
00192   return i;
00193 }
```

Here is the call graph for this function:

5.15.2.10 void xml_node_set_float (xmlNode * *node*, const xmlChar * *prop*, double *value*)

Function to set a floating point number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 305 of file utils.c.

```
00306 {
00307   xmlChar buffer[64];
00308   snprintf ((char *) buffer, 64, "%.14lg", value);
00309   xmlSetProp (node, prop, buffer);
00310 }
```

5.15.2.11 void xml_node_set_int (xmlNode * *node*, const xmlChar * *prop*, int *value*)

Function to set an integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 267 of file utils.c.

```
00268 {
00269   xmlChar buffer[64];
00270   snprintf ((char *) buffer, 64, "%d", value);
00271   xmlSetProp (node, prop, buffer);
00272 }
```

5.15.2.12 void xml_node_set_uint (xmlNode * *node*, const xmlChar * *prop*, unsigned int *value*)

Function to set an unsigned integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Unsigned integer number value.

Definition at line 286 of file utils.c.

```
00287 {
00288   xmlChar buffer[64];
00289   snprintf ((char *) buffer, 64, "%u", value);
00290   xmlSetProp (node, prop, buffer);
00291 }
```

5.16 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef UTILS__H
00039 #define UTILS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 // Public functions
00057 void show_message (char *title, char *msg, int type);
00058 void show_error (char *msg);
00059 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00060 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00061                                 int *error_code);
00062 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00063                                              const xmlChar * prop,
00064                                              unsigned int default_value,
00065                                              int *error_code);
00066 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00067                            int *error_code);
00068 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
      ,
00069                                         double default_value, int *error_code);
00070 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00071 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00072                         unsigned int value);
00073 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00074 int cores_number ();
00075 #if HAVE_GTK
```

```
00076 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00077 #endif
00078
00079 #endif
```

Index