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

```
/* config.h.  Generated from config.h.in by configure.  */
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef CONFIG__H
#define CONFIG__H 1

// Array sizes

#define MAX_NINPUTS 8
#define NALGORITHMS 3
#define NDIRECTIONS 2
#define NNORMS 4
#define NPRECISIONS 15

// Default choices

#define DEFAULT_PRECISION (NPRECISIONS - 1)
#define DEFAULT_RANDOM_SEED 7007
#define DEFAULT_RELAXATION 1.
```

```

// Interface labels

#define LOCALE_DIR "locales"
#define PROGRAM_INTERFACE "mpcotool"

// XML labels

#define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
#define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
#define XML_ADAPTATION (const xmlChar*)"adaptation"
#define XML_ALGORITHM (const xmlChar*)"algorithm"
#define XML_OPTIMIZE (const xmlChar*)"optimize"
#define XML_COORDINATES (const xmlChar*)"coordinates"
#define XML_DIRECTION (const xmlChar*)"direction"
#define XML_EUCLIDIAN (const xmlChar*)"euclidian"
#define XML_EVALUATOR (const xmlChar*)"evaluator"
#define XML_EXPERIMENT (const xmlChar*)"experiment"
#define XML_GENETIC (const xmlChar*)"genetic"
#define XML_MINIMUM (const xmlChar*)"minimum"
#define XML_MAXIMUM (const xmlChar*)"maximum"
#define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
#define XML_MUTATION (const xmlChar*)"mutation"
#define XML_NAME (const xmlChar*)"name"
#define XML_NBEST (const xmlChar*)"nbest"
#define XML_NBITS (const xmlChar*)"nbits"
#define XML_NESTIMATES (const xmlChar*)"nestimates"
#define XML_NGENERATIONS (const xmlChar*)"ngenerations"
#define XML_NITERATIONS (const xmlChar*)"niterations"
#define XML_NORM (const xmlChar*)"norm"
#define XML_NPOPULATION (const xmlChar*)"npopulation"
#define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
#define XML_NSTEPS (const xmlChar*)"nsteps"
#define XML_NSWEEPS (const xmlChar*)"nsweeps"
#define XML_P (const xmlChar*)"p"
#define XML_PRECISION (const xmlChar*)"precision"
#define XML_RANDOM (const xmlChar*)"random"
#define XML_RELAXATION (const xmlChar*)"relaxation"
#define XML_REPRODUCTION (const xmlChar*)"reproduction"
#define XML_RESULT (const xmlChar*)"result"
#define XML_SIMULATOR (const xmlChar*)"simulator"
#define XML_SEED (const xmlChar*)"seed"
#define XML_STEP (const xmlChar*)"step"
#define XML_SWEEP (const xmlChar*)"sweep"
#define XML_TAXICAB (const xmlChar*)"taxicab"
#define XML_TEMPLATE1 (const xmlChar*)"template1"
#define XML_TEMPLATE2 (const xmlChar*)"template2"
#define XML_TEMPLATE3 (const xmlChar*)"template3"
#define XML_TEMPLATE4 (const xmlChar*)"template4"
#define XML_TEMPLATE5 (const xmlChar*)"template5"
#define XML_TEMPLATE6 (const xmlChar*)"template6"
#define XML_TEMPLATE7 (const xmlChar*)"template7"
#define XML_TEMPLATE8 (const xmlChar*)"template8"
#define XML_THRESOLD (const xmlChar*)"thresold"
#define XML_TOLERANCE (const xmlChar*)"tolerance"
#define XML_VARIABLE (const xmlChar*)"variable"
#define XML_VARIABLES (const xmlChar*)"variables"
#define XML_WEIGHT (const xmlChar*)"weight"

#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);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);
  if (input->thresold != 0.)
    xml_node_set_float (node, XML_THRESOLD, input->thresold);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
      xml_node_set_float (child, XML_MINIMUM, input->rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->rangeminabs[i]);
      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.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  i = gtk_array_get_active (window->button_algorithm, NALGORITHMS);
#if DEBUG
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}
```

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.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_direction: start\n");
#endif
  i = gtk_array_get_active (window->button_direction, NDIRECTIONS);
#if DEBUG
  fprintf (stderr, "window_get_direction: %u\n", i);
  fprintf (stderr, "window_get_direction: end\n");
#endif
  return i;
}
```

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),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_direction
                                [input->direction]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->direction)
            {
            case DIRECTION_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
  gtk_spin_button_set_value (window->spin_p, input->p);
  gtk_spin_button_set_value (window->spin_thresold, input->
thresold);
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i]);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  g_signal_handler_block (window->entry_variable, window->
id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable,
input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}
```

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.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer;

#if DEBUG
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 gettext ("_Cancel"),
                                 GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_direction ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_direction ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          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.
```

```

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#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>
#ifdef G_OS_WIN32
#include <windows.h>
#elif !defined (BSD)
#include <alloca.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"
#include "interface.h"

#define DEBUG 0


#ifdef G_OS_WIN32
#define INPUT_FILE "test-ga-win.xml"
#else
#define INPUT_FILE "test-ga.xml"
#endif

const char *logo[] = {
  "32 32 3 1",
  "     c None",
  ".    c #0000FF",
  "+    c #FF0000",
  "                                ",
  "                                ",
  "                                ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .     +++     .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "    +++     .     +++    +++    ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "    +++     .      .     +++    ",
  "     .      .      .      .     ",
  "     .     +++     .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .     +++     .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "                                ",
  "                                ",
  "                                "
};

/*
const char * logo[] = {
"32 32 3 1",
"     c #FFFFFFFFFFFF",
".    c #00000000FFFF",
"X    c #FFFF00000000",
"                                ",
"                                ",
"                                ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .     XXX     .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"    XXX     .     XXX    XXX    ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"    XXX     .      .     XXX    ",
"     .      .      .      .     ",
"     .     XXX     .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .     XXX     .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"                                ",
"                                ",
"                                "};
*/

Options options[1];
Running running[1];
Window window[1];

void
input_save_direction (xmlNode * node)
{
#if DEBUG
  fprintf (stderr, "input_save_direction: start\n");
#endif
  if (input->nsteps)
    {
      xml_node_set_uint (node, XML_NSTEPS, input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        xml_node_set_float (node, XML_RELAXATION, input->relaxation);
      switch (input->direction)
        {
        case DIRECTION_METHOD_COORDINATES:
          xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
          break;
        default:
          xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
          xml_node_set_uint (node, XML_NESTIMATES, input->nestimates);
        }
    }
#if DEBUG
  fprintf (stderr, "input_save_direction: end\n");
#endif
}

void
input_save (char *filename)
```

```
{
  unsigned int i, j;
  char *buffer;
  xmlDoc *doc;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);
  file = g_file_new_for_path (input->directory);

  // Opening the input file
  doc = xmlNewDoc ((const xmlChar *) "1.0");

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp ((const xmlChar *) input->result, result_name))
    xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
  if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
    xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);
  if (input->thresold != 0.)
    xml_node_set_float (node, XML_THRESOLD, input->thresold);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
      xml_node_set_float (child, XML_MINIMUM, input->rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->rangeminabs[i]);
      xml_node_set_float (child, XML_MAXIMUM, input->rangemax[i]);
      if (input->rangemaxabs[i] != G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->rangemaxabs[i]);
      if (input->precision[i] != DEFAULT_PRECISION)
        xml_node_set_uint (child, XML_PRECISION, input->precision[i]);
      if (input->algorithm == ALGORITHM_SWEEP)
        xml_node_set_uint (child, XML_NSWEEPS, input->nsweeps[i]);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, XML_NBITS, input->nbits[i]);
      if (input->nsteps)
        xml_node_set_float (child, XML_STEP, input->step[i]);
    }

  // Saving the error norm
  switch (input->norm)
    {
    case ERROR_NORM_MAXIMUM:
      xmlSetProp (node, XML_NORM, XML_MAXIMUM);
      break;
    case ERROR_NORM_P:
      xmlSetProp (node, XML_NORM, XML_P);
      xml_node_set_float (node, XML_P, input->p);
      break;
    case ERROR_NORM_TAXICAB:
      xmlSetProp (node, XML_NORM, XML_TAXICAB);
    }

  // Saving the XML file
  xmlSaveFormatFile (filename, doc, 1);

  // Freeing memory
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_save: end\n");
#endif
}

void
options_new ()
{
#if DEBUG
  fprintf (stderr, "options_new: start\n");
#endif
  options->label_seed = (GtkLabel *)
    gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
  options->spin_seed = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_seed),
     gettext ("Seed to init the pseudo-random numbers generator"));
  gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
  options->label_threads = (GtkLabel *)
```

```
    gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
  options->spin_threads
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_threads),
     gettext ("Number of threads to perform the calibration/optimization for "
              "the stochastic algorithm"));
  gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
  options->label_direction = (GtkLabel *)
    gtk_label_new (gettext ("Threads number for the direction search method"));
  options->spin_direction
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_direction),
     gettext ("Number of threads to perform the calibration/optimization for "
              "the direction search method"));
  gtk_spin_button_set_value (options->spin_direction,
                             (gdouble) nthreads_direction);
  options->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
                   0, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
                   1, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
                   0, 2, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
                   1, 2, 1, 1);
  gtk_widget_show_all (GTK_WIDGET (options->grid));
  options->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (gettext ("Options"),
                                 window->window,
                                 GTK_DIALOG_MODAL,
                                 gettext ("_OK"), GTK_RESPONSE_OK,
                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
                                 NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
     GTK_WIDGET (options->grid));
  if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
    {
      input->seed
        = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
      nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
      nthreads_direction
        = gtk_spin_button_get_value_as_int (options->spin_direction);
    }
  gtk_widget_destroy (GTK_WIDGET (options->dialog));
#if DEBUG
  fprintf (stderr, "options_new: end\n");
#endif
}

void
running_new ()
{
#if DEBUG
  fprintf (stderr, "running_new: start\n");
#endif
  running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
  running->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (gettext ("Calculating"),
                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
     GTK_WIDGET (running->label));
  gtk_widget_show_all (GTK_WIDGET (running->dialog));
#if DEBUG
  fprintf (stderr, "running_new: end\n");
#endif
}

unsigned int
window_get_algorithm ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  i = gtk_array_get_active (window->button_algorithm, NALGORITHMS);
#if DEBUG
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
```

```
  return i;
}

unsigned int
window_get_direction ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_direction: start\n");
#endif
  i = gtk_array_get_active (window->button_direction, NDIRECTIONS);
#if DEBUG
  fprintf (stderr, "window_get_direction: %u\n", i);
  fprintf (stderr, "window_get_direction: end\n");
#endif
  return i;
}

unsigned int
window_get_norm ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_norm: start\n");
#endif
  i = gtk_array_get_active (window->button_norm, NNORMS);
#if DEBUG
  fprintf (stderr, "window_get_norm: %u\n", i);
  fprintf (stderr, "window_get_norm: end\n");
#endif
  return i;
}

void
window_save_direction ()
{
#if DEBUG
  fprintf (stderr, "window_save_direction: start\n");
#endif
  if (gtk_toggle_button_get_active
      (GTK_TOGGLE_BUTTON (window->check_direction)))
    {
      input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
      input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
      switch (window_get_direction ())
        {
        case DIRECTION_METHOD_COORDINATES:
          input->direction = DIRECTION_METHOD_COORDINATES;
          break;
        default:
          input->direction = DIRECTION_METHOD_RANDOM;
          input->nestimates
            = gtk_spin_button_get_value_as_int (window->spin_estimates);
        }
    }
  else
    input->nsteps = 0;
#if DEBUG
  fprintf (stderr, "window_save_direction: end\n");
#endif
}

int
window_save ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer;

#if DEBUG
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 gettext ("_Cancel"),
                                 GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
          window_save_direction ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
          window_save_direction ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }
      input->norm = window_get_norm ();
      input->p = gtk_spin_button_get_value (window->spin_p);
      input->thresold = gtk_spin_button_get_value (window->spin_thresold);

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}

void
window_run ()
{
  unsigned int i;
  char *msg, *msg2, buffer[64], buffer2[64];
#if DEBUG
  fprintf (stderr, "window_run: start\n");
#endif
  if (!window_save ())
    {
#if DEBUG
      fprintf (stderr, "window_run: end\n");
#endif
      return;
    }
  running_new ();
  while (gtk_events_pending ())
    gtk_main_iteration ();
  optimize_open ();
  gtk_widget_destroy (GTK_WIDGET (running->dialog));
  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
  msg2 = g_strdup (buffer);
  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
    {
      snprintf (buffer, 64, "%s = %s\n",
                optimize->label[i], format[optimize->
precision[i]]);
      snprintf (buffer2, 64, buffer, optimize->value_old[i]);
      msg = g_strconcat (msg2, buffer2, NULL);
      g_free (msg2);
    }
  snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
            optimize->calculation_time);
  msg = g_strconcat (msg2, buffer, NULL);
  g_free (msg2);
  show_message (gettext ("Best result"), msg, INFO_TYPE);
  g_free (msg);
  optimize_free ();
#if DEBUG
  fprintf (stderr, "window_run: end\n");
#endif
}

void
window_help ()
{
  char *buffer, *buffer2;
#if DEBUG
  fprintf (stderr, "window_help: start\n");
#endif
  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
                              gettext ("user-manual.pdf"), NULL);
  buffer = g_filename_to_uri (buffer2, NULL, NULL);
  g_free (buffer2);
  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
#if DEBUG
  fprintf (stderr, "window_help: uri=%s\n", buffer);
#endif
  g_free (buffer);
#if DEBUG
  fprintf (stderr, "window_help: end\n");
#endif
}

void
window_about ()
{
  static const gchar *authors[] = {
    "Javier Burguete Tolosa <jburguete@eead.csic.es>",
    "Borja Latorre Garcés <borja.latorre@csic.es>",
    NULL
  };
#if DEBUG
  fprintf (stderr, "window_about: start\n");
#endif
  gtk_show_about_dialog
    (window->window,
     "program_name", "MPCOTool",
     "comments",
     gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
              "A software to perform calibrations or optimizations of "
              "empirical parameters"),
     "authors", authors,
     "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
     "version", "2.0.1",
     "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
     "logo", window->logo,
     "website", "https://github.com/jburguete/mpcotool",
     "license-type", GTK_LICENSE_BSD, NULL);
#if DEBUG
  fprintf (stderr, "window_about: end\n");
#endif
}

void
window_update_direction ()
{
#if DEBUG
  fprintf (stderr, "window_update_direction: start\n");
#endif
  gtk_widget_show (GTK_WIDGET (window->check_direction));
  if (gtk_toggle_button_get_active
      (GTK_TOGGLE_BUTTON (window->check_direction)))
    {
      gtk_widget_show (GTK_WIDGET (window->grid_direction));
      gtk_widget_show (GTK_WIDGET (window->label_step));
      gtk_widget_show (GTK_WIDGET (window->spin_step));
    }
  switch (window_get_direction ())
    {
    case DIRECTION_METHOD_COORDINATES:
      gtk_widget_hide (GTK_WIDGET (window->label_estimates));
      gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_estimates));
      gtk_widget_show (GTK_WIDGET (window->spin_estimates));
    }
#if DEBUG
  fprintf (stderr, "window_update_direction: end\n");
#endif
}

void
window_update ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_update: start\n");
#endif
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_evaluator),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                   (window->check_evaluator)));
  gtk_widget_hide (GTK_WIDGET (window->label_simulations));
  gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
  gtk_widget_hide (GTK_WIDGET (window->label_iterations));
  gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
  gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->label_bests));
  gtk_widget_hide (GTK_WIDGET (window->spin_bests));
  gtk_widget_hide (GTK_WIDGET (window->label_population));
  gtk_widget_hide (GTK_WIDGET (window->spin_population));
  gtk_widget_hide (GTK_WIDGET (window->label_generations));
  gtk_widget_hide (GTK_WIDGET (window->spin_generations));
  gtk_widget_hide (GTK_WIDGET (window->label_mutation));
  gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
  gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->label_bits));
  gtk_widget_hide (GTK_WIDGET (window->spin_bits));
  gtk_widget_hide (GTK_WIDGET (window->check_direction));
  gtk_widget_hide (GTK_WIDGET (window->grid_direction));
  gtk_widget_hide (GTK_WIDGET (window->label_step));
  gtk_widget_hide (GTK_WIDGET (window->spin_step));
  gtk_widget_hide (GTK_WIDGET (window->label_p));
  gtk_widget_hide (GTK_WIDGET (window->spin_p));
  i = gtk_spin_button_get_value_as_int (window->spin_iterations);
  switch (window_get_algorithm ())
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_widget_show (GTK_WIDGET (window->label_simulations));
      gtk_widget_show (GTK_WIDGET (window->spin_simulations));
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      window_update_direction ();
      break;
    case ALGORITHM_SWEEP:
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      gtk_widget_show (GTK_WIDGET (window->label_sweeps));
      gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
      gtk_widget_show (GTK_WIDGET (window->check_direction));
      window_update_direction ();
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_population));
      gtk_widget_show (GTK_WIDGET (window->spin_population));
      gtk_widget_show (GTK_WIDGET (window->label_generations));
      gtk_widget_show (GTK_WIDGET (window->spin_generations));
      gtk_widget_show (GTK_WIDGET (window->label_mutation));
      gtk_widget_show (GTK_WIDGET (window->spin_mutation));
      gtk_widget_show (GTK_WIDGET (window->label_reproduction));
      gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
      gtk_widget_show (GTK_WIDGET (window->label_adaptation));
      gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
      gtk_widget_show (GTK_WIDGET (window->label_bits));
      gtk_widget_show (GTK_WIDGET (window->spin_bits));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
  for (i = 0; i < input->ninputs; ++i)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->id_input[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
      g_signal_handler_unblock
        (window->button_template[i], window->id_input[i]);
      g_signal_handler_unblock
        (window->check_template[i], window->id_template[i]);
    }
  if (i > 0)
    {
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i - 1]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
    }
  if (i < MAX_NINPUTS)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i]));
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->id_input[i]);
      gtk_toggle_button_set_active
```

```
        (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
      g_signal_handler_unblock
        (window->button_template[i], window->id_input[i]);
      g_signal_handler_unblock
        (window->check_template[i], window->id_template[i]);
    }
  while (++i < MAX_NINPUTS)
    {
      gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
      gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_minabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_maxabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
  if (window_get_norm () == ERROR_NORM_P)
    {
      gtk_widget_show (GTK_WIDGET (window->label_p));
      gtk_widget_show (GTK_WIDGET (window->spin_p));
    }
#if DEBUG
  fprintf (stderr, "window_update: end\n");
#endif
}

void
window_set_algorithm ()
{
  int i;
#if DEBUG
  fprintf (stderr, "window_set_algorithm: start\n");
#endif
  i = window_get_algorithm ();
  switch (i)
    {
    case ALGORITHM_SWEEP:
      input->nsweeps = (unsigned int *) g_realloc
        (input->nsweeps, input->nvariables * sizeof (unsigned int));
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->nsweeps[i]);
      break;
    case ALGORITHM_GENETIC:
      input->nbits = (unsigned int *) g_realloc
        (input->nbits, input->nvariables * sizeof (unsigned int));
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
nbits[i]);
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_set_algorithm: end\n");
#endif
}

void
window_set_experiment ()
{
  unsigned int i, j;
  char *buffer1, *buffer2;
#if DEBUG
  fprintf (stderr, "window_set_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
  buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
  buffer2 = g_build_filename (input->directory, buffer1, NULL);
  g_free (buffer1);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_file_chooser_set_filename
    (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_free (buffer2);
  for (j = 0; j < input->ninputs; ++j)
    {
      g_signal_handler_block (window->button_template[j], window->
id_input[j]);
      buffer2
        = g_build_filename (input->directory, input->template[j][i], NULL);
      gtk_file_chooser_set_filename
        (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
      g_free (buffer2);
      g_signal_handler_unblock
        (window->button_template[j], window->id_input[j]);
    }
#if DEBUG
  fprintf (stderr, "window_set_experiment: end\n");
#endif
}

void
window_remove_experiment ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_remove_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  xmlFree (input->experiment[i]);
  --input->nexperiments;
  for (j = i; j < input->nexperiments; ++j)
    {
      input->experiment[j] = input->experiment[j + 1];
      input->weight[j] = input->weight[j + 1];
    }
  j = input->nexperiments - 1;
  if (i > j)
    i = j;
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_remove_experiment: end\n");
#endif
}

void
window_add_experiment ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_add_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_insert_text
    (window->combo_experiment, i, input->experiment[i]);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  input->experiment = (char **) g_realloc
    (input->experiment, (input->nexperiments + 1) * sizeof (char *));
  input->weight = (double *) g_realloc
    (input->weight, (input->nexperiments + 1) * sizeof (double));
  for (j = input->nexperiments - 1; j > i; --j)
    {
      input->experiment[j + 1] = input->experiment[j];
      input->weight[j + 1] = input->weight[j];
    }
  input->experiment[j + 1]
    = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
  input->weight[j + 1] = input->weight[j];
  ++input->nexperiments;
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->ninputs; ++j)
```

```
    g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_add_experiment: end\n");
#endif
}

void
window_name_experiment ()
{
  unsigned int i;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_name_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_name_experiment: end\n");
#endif
}

void
window_weight_experiment ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_weight_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
#if DEBUG
  fprintf (stderr, "window_weight_experiment: end\n");
#endif
}

void
window_inputs_experiment ()
{
  unsigned int j;
#if DEBUG
  fprintf (stderr, "window_inputs_experiment: start\n");
#endif
  j = input->ninputs - 1;
  if (j
      && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                        (window->check_template[j])))
    --input->ninputs;
  if (input->ninputs < MAX_NINPUTS
      && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                       (window->check_template[j])))
    {
      ++input->ninputs;
      for (j = 0; j < input->ninputs; ++j)
        {
          input->template[j] = (char **)
            g_realloc (input->template[j], input->nvariables * sizeof (char *))
;
        }
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_inputs_experiment: end\n");
#endif
}

void
window_template_experiment (void *data)
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
```

```
#if DEBUG
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}

void
window_set_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_set_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_entry_set_text (window->entry_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
  gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
  if (input->rangeminabs[i] != -G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_minabs, input->rangeminabs[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
    }
  if (input->rangemaxabs[i] != G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_maxabs, input->rangemaxabs[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
    }
  gtk_spin_button_set_value (window->spin_precision, input->precision[i]);
  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
  if (input->nsteps)
    gtk_spin_button_set_value (window->spin_step, input->step[i]);
#if DEBUG
  fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
           input->precision[i]);
#endif
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->nsweeps[i]);
#if DEBUG
      fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
               input->nsweeps[i]);
#endif
      break;
    case ALGORITHM_GENETIC:
      gtk_spin_button_set_value (window->spin_bits, (gdouble) input->nbits[i]);
#if DEBUG
      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
               input->nbits[i]);
#endif
      break;
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_set_variable: end\n");
#endif
}

void
window_remove_variable ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_remove_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  xmlFree (input->label[i]);
  --input->nvariables;
  for (j = i; j < input->nvariables; ++j)
    {
      input->label[j] = input->label[j + 1];
      input->rangemin[j] = input->rangemin[j + 1];
      input->rangemax[j] = input->rangemax[j + 1];
      input->rangeminabs[j] = input->rangeminabs[j + 1];
      input->rangemaxabs[j] = input->rangemaxabs[j + 1];
      input->precision[j] = input->precision[j + 1];
      input->step[j] = input->step[j + 1];
      switch (window_get_algorithm ())
        {
        case ALGORITHM_SWEEP:
          input->nsweeps[j] = input->nsweeps[j + 1];
          break;
        case ALGORITHM_GENETIC:
          input->nbits[j] = input->nbits[j + 1];
        }
    }
  j = input->nvariables - 1;
  if (i > j)
    i = j;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_remove_variable: end\n");
#endif
}

void
window_add_variable ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_add_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_insert_text (window->combo_variable, i, input->label[i]);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  input->label = (char **) g_realloc
    (input->label, (input->nvariables + 1) * sizeof (char *));
  input->rangemin = (double *) g_realloc
    (input->rangemin, (input->nvariables + 1) * sizeof (double));
  input->rangemax = (double *) g_realloc
    (input->rangemax, (input->nvariables + 1) * sizeof (double));
  input->rangeminabs = (double *) g_realloc
    (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
  input->rangemaxabs = (double *) g_realloc
    (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
  input->precision = (unsigned int *) g_realloc
    (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
  input->step = (double *) g_realloc
    (input->step, (input->nvariables + 1) * sizeof (double));
  for (j = input->nvariables - 1; j > i; --j)
    {
      input->label[j + 1] = input->label[j];
      input->rangemin[j + 1] = input->rangemin[j];
      input->rangemax[j + 1] = input->rangemax[j];
      input->rangeminabs[j + 1] = input->rangeminabs[j];
      input->rangemaxabs[j + 1] = input->rangemaxabs[j];
      input->precision[j + 1] = input->precision[j];
      input->step[j + 1] = input->step[j];
    }
  input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
  input->rangemin[j + 1] = input->rangemin[j];
  input->rangemax[j + 1] = input->rangemax[j];
  input->rangeminabs[j + 1] = input->rangeminabs[j];
  input->rangemaxabs[j + 1] = input->rangemaxabs[j];
  input->precision[j + 1] = input->precision[j];
  input->step[j + 1] = input->step[j];
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      input->nsweeps = (unsigned int *) g_realloc
        (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
      for (j = input->nvariables - 1; j > i; --j)
        input->nsweeps[j + 1] = input->nsweeps[j];
      input->nsweeps[j + 1] = input->nsweeps[j];
      break;
    case ALGORITHM_GENETIC:
      input->nbits = (unsigned int *) g_realloc
        (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
      for (j = input->nvariables - 1; j > i; --j)
        input->nbits[j + 1] = input->nbits[j];
      input->nbits[j + 1] = input->nbits[j];
    }
  ++input->nvariables;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_add_variable: end\n");
#endif
}

void
window_label_variable ()
{
  unsigned int i;
  const char *buffer;
#if DEBUG
  fprintf (stderr, "window_label_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  buffer = gtk_entry_get_text (window->entry_variable);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
#if DEBUG
  fprintf (stderr, "window_label_variable: end\n");
#endif
}

void
window_precision_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_precision_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->precision[i]
    = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
  gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
#if DEBUG
  fprintf (stderr, "window_precision_variable: end\n");
#endif
}

void
window_rangemin_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemin_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
#if DEBUG
  fprintf (stderr, "window_rangemin_variable: end\n");
#endif
}

void
window_rangemax_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemax_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
#if DEBUG
  fprintf (stderr, "window_rangemax_variable: end\n");
#endif
}

void
window_rangeminabs_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangeminabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangeminabs[i] = gtk_spin_button_get_value (window->spin_minabs);
#if DEBUG
  fprintf (stderr, "window_rangeminabs_variable: end\n");
#endif
}

void
window_rangemaxabs_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemaxabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemaxabs[i] = gtk_spin_button_get_value (window->spin_maxabs);
#if DEBUG
  fprintf (stderr, "window_rangemaxabs_variable: end\n");
#endif
}

void
window_step_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_step_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->step[i] = gtk_spin_button_get_value (window->spin_step);
#if DEBUG
  fprintf (stderr, "window_step_variable: end\n");
#endif
}

void
window_update_variable ()
{
  int i;
#if DEBUG
  fprintf (stderr, "window_update_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  if (i < 0)
    i = 0;
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      input->nsweeps[i]
        = gtk_spin_button_get_value_as_int (window->spin_sweeps);
#if DEBUG
      fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
               input->nsweeps[i]);
```

```
#endif
      break;
    case ALGORITHM_GENETIC:
      input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
#if DEBUG
      fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
               input->nbits[i]);
#endif
    }
#if DEBUG
  fprintf (stderr, "window_update_variable: end\n");
#endif
}

int
window_read (char *filename)
{
  unsigned int i;
  char *buffer;
#if DEBUG
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  if (!input_open (filename))
    return 0;

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_direction
                                [input->direction]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->direction)
            {
            case DIRECTION_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
  gtk_spin_button_set_value (window->spin_p, input->p);
  gtk_spin_button_set_value (window->spin_thresold, input->thresold);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i]);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}

void
window_open ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer, *directory, *name;

#if DEBUG
  fprintf (stderr, "window_open: start\n");
#endif

  // Saving a backup of the current input file
  directory = g_strdup (input->directory);
  name = g_strdup (input->name);

  // Opening dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Open input file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_OPEN,
                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK saving
  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Traying to open the input file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      if (!window_read (buffer))
        {
#if DEBUG
          fprintf (stderr, "window_open: error reading input file\n");
#endif
          g_free (buffer);

          // Reading backup file on error
          buffer = g_build_filename (directory, name, NULL);
          if (!input_open (buffer))
            {

              // Closing on backup file reading error
#if DEBUG
              fprintf (stderr, "window_read: error reading backup file\n");
#endif
              g_free (buffer);
              break;
            }
          g_free (buffer);
        }
      else
        {
          g_free (buffer);
          break;
        }
    }

  // Freeing and closing
  g_free (name);
  g_free (directory);
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_open: end\n");
#endif
}

void
window_new ()
{
  unsigned int i;
  char *buffer, *buffer2, buffer3[64];
  char *label_algorithm[NALGORITHMS] = {
    "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
  };
  char *tip_algorithm[NALGORITHMS] = {
    gettext ("Monte-Carlo brute force algorithm"),
    gettext ("Sweep brute force algorithm"),
    gettext ("Genetic algorithm")
  };
  char *label_direction[NDIRECTIONS] = {
    gettext ("_Coordinates descent"), gettext ("_Random")
  };
  char *tip_direction[NDIRECTIONS] = {
    gettext ("Coordinates direction estimate method"),
    gettext ("Random direction estimate method")
  };
  char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
  char *tip_norm[NNORMS] = {
    gettext ("Euclidean error norm (L2)"),
    gettext ("Maximum error norm (L)"),
    gettext ("P error norm (Lp)"),
    gettext ("Taxicab error norm (L1)")
  };

#if DEBUG
  fprintf (stderr, "window_new: start\n");
#endif

  // Creating the window
  window->window = main_window
    = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);

  // Finish when closing the window
  g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);

  // Setting the window title
  gtk_window_set_title (window->window, "MPCOTool");

  // Creating the open button
  window->button_open = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-open",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Open"));
  g_signal_connect (window->button_open, "clicked", window_open, NULL);

  // Creating the save button
  window->button_save = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-save",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Save"));
  g_signal_connect (window->button_save, "clicked", (void (*)) window_save,
                    NULL);

  // Creating the run button
  window->button_run = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("system-run",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Run"));
  g_signal_connect (window->button_run, "clicked", window_run, NULL);

  // Creating the options button
  window->button_options = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("preferences-system",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Options"));
  g_signal_connect (window->button_options, "clicked", options_new, NULL);

  // Creating the help button
  window->button_help = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-browser",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Help"));
  g_signal_connect (window->button_help, "clicked", window_help, NULL);

  // Creating the about button
  window->button_about = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-about",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("About"));
  g_signal_connect (window->button_about, "clicked", window_about, NULL);

  // Creating the exit button
  window->button_exit = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("application-exit",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Exit"));
  g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);

  // Creating the buttons bar
  window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
  gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);

  // Creating the simulator program label and entry
  window->label_simulator
    = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
  window->button_simulator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Simulator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
                               gettext ("Simulator program executable file"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);

  // Creating the evaluator program label and entry
  window->check_evaluator = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
  g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
  window->button_evaluator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Evaluator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_evaluator),
     gettext ("Optional evaluator program executable file"));

  // Creating the results files labels and entries
  window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
  window->entry_result = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
  window->label_variables
    = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
  window->entry_variables = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variables),
     gettext ("All simulated results file"));

  // Creating the files grid and attaching widgets
  window->grid_files = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
                   0, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
                   1, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
                   0, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
                   1, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
                   0, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
                   1, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
                   0, 3, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
                   1, 3, 1, 1);

  // Creating the algorithm properties
  window->label_simulations = (GtkLabel *) gtk_label_new
    (gettext ("Simulations number"));
  window->spin_simulations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_simulations),
     gettext ("Number of simulations to perform for each iteration"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
  window->label_iterations = (GtkLabel *)
    gtk_label_new (gettext ("Iterations number"));
  window->spin_iterations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
  g_signal_connect
    (window->spin_iterations, "value-changed", window_update, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
  window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
  window->spin_tolerance
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_tolerance),
     gettext ("Tolerance to set the variable interval on the next iteration"));
  window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
  window->spin_bests
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bests),
     gettext ("Number of best simulations used to set the variable interval "
              "on the next iteration"));
  window->label_population
    = (GtkLabel *) gtk_label_new (gettext ("Population number"));
  window->spin_population
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_population),
     gettext ("Number of population for the genetic algorithm"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
  window->label_generations
    = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
  window->spin_generations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_generations),
     gettext ("Number of generations for the genetic algorithm"));
  window->label_mutation
    = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
  window->spin_mutation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_mutation),
     gettext ("Ratio of mutation for the genetic algorithm"));
  window->label_reproduction
    = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
  window->spin_reproduction
```

```
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_reproduction),
     gettext ("Ratio of reproduction for the genetic algorithm"));
  window->label_adaptation
    = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
  window->spin_adaptation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_adaptation),
     gettext ("Ratio of adaptation for the genetic algorithm"));
  window->label_thresold = (GtkLabel *) gtk_label_new (gettext ("Thresold"));
  window->spin_thresold = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_thresold),
     gettext ("Thresold in the objective function to finish the simulations"));
  window->scrolled_thresold
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_thresold),
                     GTK_WIDGET (window->spin_thresold));
//  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_thresold), TRUE);
//  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_thresold),
//                               GTK_ALIGN_FILL);

  // Creating the direction search method properties
  window->check_direction = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
  g_signal_connect (window->check_direction, "clicked", window_update, NULL);
  window->grid_direction = (GtkGrid *) gtk_grid_new ();
  window->button_direction[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
  gtk_grid_attach (window->grid_direction,
                   GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_direction[0], "clicked", window_update,
                    NULL);
  for (i = 0; ++i < NDIRECTIONS;)
    {
      window->button_direction[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_direction[0]),
         label_direction[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
                                   tip_direction[i]);
      gtk_grid_attach (window->grid_direction,
                       GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
      g_signal_connect (window->button_direction[i], "clicked",
                        window_update, NULL);
    }
  window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
  window->spin_steps = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
  window->label_estimates
    = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
  window->spin_estimates = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e3, 1.);
  window->label_relaxation
    = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
  window->spin_relaxation = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., 2., 0.001);
  gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->label_steps),
                   0, NDIRECTIONS, 1, 1);
  gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->spin_steps),
                   1, NDIRECTIONS, 1, 1);
  gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->label_estimates),
                   0, NDIRECTIONS + 1, 1, 1);
  gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->spin_estimates),
                   1, NDIRECTIONS + 1, 1, 1);
  gtk_grid_attach (window->grid_direction,
                   GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
                   1);
  gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->spin_relaxation),
                   1, NDIRECTIONS + 2, 1, 1);

  // Creating the array of algorithms
  window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
  window->button_algorithm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
                              tip_algorithm[0]);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_algorithm[0], "clicked",
                    window_set_algorithm, NULL);
  for (i = 0; ++i < NALGORITHMS;)
    {
      window->button_algorithm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_algorithm[0]),
         label_algorithm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
                                   tip_algorithm[i]);
      gtk_grid_attach (window->grid_algorithm,
                       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_algorithm[i], "clicked",
                        window_set_algorithm, NULL);
    }
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_simulations), 0,
                   NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_iterations), 0,
                   NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_iterations), 1,
                   NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_tolerance), 0,
                   NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_tolerance), 1,
                   NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_population), 0,
                   NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_population), 1,
                   NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_generations), 0,
                   NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_generations), 1,
                   NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_mutation), 0,
                   NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_mutation), 1,
                   NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_reproduction), 0,
                   NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_reproduction), 1,
                   NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_adaptation), 0,
                   NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_adaptation), 1,
                   NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->check_direction), 0,
                   NALGORITHMS + 9, 2, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->grid_direction), 0,
                   NALGORITHMS + 10, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_thresold),
                   0, NALGORITHMS + 11, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->scrolled_thresold), 1,
                   NALGORITHMS + 11, 1, 1);
  window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
  gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
                     GTK_WIDGET (window->grid_algorithm));

  // Creating the variable widgets
  window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
  window->id_variable = g_signal_connect
    (window->combo_variable, "changed", window_set_variable, NULL);
  window->button_add_variable
    = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_variable, "clicked",
window_add_variable, NULL);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
  window->button_remove_variable
    = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
  window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
  window->entry_variable = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
  window->id_variable_label = g_signal_connect
    (window->entry_variable, "changed", window_label_variable, NULL);
  window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
  window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_min),
     gettext ("Minimum initial value of the variable"));
  window->scrolled_min
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_min),
                     GTK_WIDGET (window->spin_min));
  g_signal_connect (window->spin_min, "value-changed",
                    window_rangemin_variable, NULL);
  window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
  window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_max),
     gettext ("Maximum initial value of the variable"));
  window->scrolled_max
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_max),
                     GTK_WIDGET (window->spin_max));
  g_signal_connect (window->spin_max, "value-changed",
                    window_rangemax_variable, NULL);
  window->check_minabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
  g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
  window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_minabs),
     gettext ("Minimum allowed value of the variable"));
  window->scrolled_minabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
                     GTK_WIDGET (window->spin_minabs));
  g_signal_connect (window->spin_minabs, "value-changed",
                    window_rangeminabs_variable, NULL);
  window->check_maxabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
  g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
  window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_maxabs),
     gettext ("Maximum allowed value of the variable"));
  window->scrolled_maxabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
                     GTK_WIDGET (window->spin_maxabs));
  g_signal_connect (window->spin_maxabs, "value-changed",
                    window_rangemaxabs_variable, NULL);
  window->label_precision
    = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
  window->spin_precision = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_precision),
     gettext ("Number of precision floating point digits\n"
              "0 is for integer numbers"));
  g_signal_connect (window->spin_precision, "value-changed",
                    window_precision_variable, NULL);
  window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
  window->spin_sweeps
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_sweeps),
     gettext ("Number of steps sweeping the variable"));
  g_signal_connect
    (window->spin_sweeps, "value-changed", window_update_variable, NULL);
  window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
  window->spin_bits
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bits),
     gettext ("Number of bits to encode the variable"));
  g_signal_connect
    (window->spin_bits, "value-changed", window_update_variable, NULL);
  window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
  window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_step),
     gettext ("Initial step size for the direction search method"));
  window->scrolled_step
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_step),
                     GTK_WIDGET (window->spin_step));
  g_signal_connect
    (window->spin_step, "value-changed", window_step_variable, NULL);
  window->grid_variable = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
  window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
  gtk_container_add (GTK_CONTAINER (window->frame_variable),
                     GTK_WIDGET (window->grid_variable));

  // Creating the experiment widgets
  window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
                               gettext ("Experiment selector"));
  window->id_experiment = g_signal_connect
    (window->combo_experiment, "changed", window_set_experiment, NULL)
;
  window->button_add_experiment
    = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
                               gettext ("Add experiment"));
  window->button_remove_experiment
    = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_experiment, "clicked",
                    window_remove_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
                               gettext ("Remove experiment"));
  window->label_experiment
    = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
  window->button_experiment = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Experimental data file"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
                               gettext ("Experimental data file"));
  window->id_experiment_name
    = g_signal_connect (window->button_experiment, "selection-changed",
                        window_name_experiment, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
  window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
  window->spin_weight
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_weight),
     gettext ("Weight factor to build the objective function"));
  g_signal_connect
    (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
  window->grid_experiment = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
  for (i = 0; i < MAX_NINPUTS; ++i)
    {
      snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
      window->check_template[i] = (GtkCheckButton *)
        gtk_check_button_new_with_label (buffer3);
      window->id_template[i]
        = g_signal_connect (window->check_template[i], "toggled",
                            window_inputs_experiment, NULL);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
      window->button_template[i] = (GtkFileChooserButton *)
        gtk_file_chooser_button_new (gettext ("Input template"),
                                     GTK_FILE_CHOOSER_ACTION_OPEN);
      gtk_widget_set_tooltip_text
        (GTK_WIDGET (window->button_template[i]),
         gettext ("Experimental input template file"));
      window->id_input[i]
        = g_signal_connect_swapped (window->button_template[i],
                                    "selection-changed",
                                    (void (*)) window_template_experiment,
                                    (void *) (size_t) i);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
    }
  window->frame_experiment
    = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
  gtk_container_add (GTK_CONTAINER (window->frame_experiment),
                     GTK_WIDGET (window->grid_experiment));

  // Creating the error norm widgets
  window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
  window->grid_norm = (GtkGrid *) gtk_grid_new ();
  gtk_container_add (GTK_CONTAINER (window->frame_norm),
                     GTK_WIDGET (window->grid_norm));
  window->button_norm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
                               tip_norm[0]);
  gtk_grid_attach (window->grid_norm,
                   GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
  for (i = 0; ++i < NNORMS;)
    {
      window->button_norm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
                                   tip_norm[i]);
      gtk_grid_attach (window->grid_norm,
                       GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
    }
  window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
  window->spin_p = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
  window->scrolled_p
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_p),
                     GTK_WIDGET (window->spin_p));
  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
                   1, 2, 1, 2);

  // Creating the grid and attaching the widgets to the grid
  window->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));

  // Setting the window logo
  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
  gtk_window_set_icon (window->window, window->logo);

  // Showing the window
  gtk_widget_show_all (GTK_WIDGET (window->window));

  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
#if GTK_MINOR_VERSION >= 16
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_thresold), -1, 40);
#endif

  // Reading initial example
  input_new ();
  buffer2 = g_get_current_dir ();
  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
  g_free (buffer2);
  window_read (buffer);
  g_free (buffer);

#if DEBUG
  fprintf (stderr, "window_new: start\n");
#endif
}
```

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

Copyright 2012-2016, all rights reserved.

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.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

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.

```
{
  unsigned int i, j;
  char *buffer;
  xmlDoc *doc;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);
  file = g_file_new_for_path (input->directory);

  // Opening the input file
  doc = xmlNewDoc ((const xmlChar *) "1.0");

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp ((const xmlChar *) input->result, result_name))
    xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
  if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
    xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_direction (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);
  if (input->thresold != 0.)
    xml_node_set_float (node, XML_THRESOLD, input->thresold);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        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.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_direction: start\n");
#endif
  i = gtk_array_get_active (window->button_direction, NDIRECTIONS);
#if DEBUG
  fprintf (stderr, "window_get_direction: %u\n", i);
  fprintf (stderr, "window_get_direction: end\n");
#endif
  return i;
}
```

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.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_norm: start\n");
#endif
  i = gtk_array_get_active (window->button_norm, NNORMS);
#if DEBUG
  fprintf (stderr, "window_get_norm: %u\n", i);
  fprintf (stderr, "window_get_norm: end\n");
#endif
  return i;
}
```

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.

```
{
  unsigned int i;
  char *buffer;
#if DEBUG
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  if (!input_open (filename))
    return 0;

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_direction
                                [input->direction]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->direction)
            {
            case DIRECTION_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 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
```

```
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_direction ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_direction ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }
      input->norm = window_get_norm ();
      input->p = gtk_spin_button_get_value (window->spin_p);
      input->thresold = gtk_spin_button_get_value (window->
spin_thresold);

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_save: end\n");
```

```
#endif
  return 0;
}
```

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.

```
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}
```

5.6 interface.h

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef INTERFACE__H
#define INTERFACE__H 1

#define MAX_LENGTH (DEFAULT_PRECISION + 8)

typedef struct
```

```
{
  char *template[MAX_NINPUTS];
  char *name;
  double weight;
} Experiment;

typedef struct
{
  char *label;
  double rangemin;
  double rangemax;
  double rangeminabs;
  double rangemaxabs;
  double step;
  unsigned int precision;
  unsigned int nsweeps;
  unsigned int nbits;
} Variable;

typedef struct
{
  GtkDialog *dialog;
  GtkGrid *grid;
  GtkLabel *label_seed;
  GtkSpinButton *spin_seed;
  GtkLabel *label_threads;
  GtkSpinButton *spin_threads;
  GtkLabel *label_direction;
  GtkSpinButton *spin_direction;
} Options;

typedef struct
{
  GtkDialog *dialog;
  GtkLabel *label;
} Running;

typedef struct
{
  GtkWindow *window;
  GtkGrid *grid;
  GtkToolbar *bar_buttons;
  GtkToolButton *button_open;
  GtkToolButton *button_save;
  GtkToolButton *button_run;
  GtkToolButton *button_options;
  GtkToolButton *button_help;
  GtkToolButton *button_about;
  GtkToolButton *button_exit;
  GtkGrid *grid_files;
  GtkLabel *label_simulator;
  GtkFileChooserButton *button_simulator;
  GtkCheckButton *check_evaluator;
  GtkFileChooserButton *button_evaluator;
  GtkLabel *label_result;
  GtkEntry *entry_result;
  GtkLabel *label_variables;
  GtkEntry *entry_variables;
  GtkFrame *frame_norm;
  GtkGrid *grid_norm;
  GtkRadioButton *button_norm[NNORMS];
  GtkLabel *label_p;
  GtkSpinButton *spin_p;
  GtkScrolledWindow *scrolled_p;
  GtkFrame *frame_algorithm;
  GtkGrid *grid_algorithm;
  GtkRadioButton *button_algorithm[NALGORITHMS];
  GtkLabel *label_simulations;
  GtkSpinButton *spin_simulations;
  GtkLabel *label_iterations;
  GtkSpinButton *spin_iterations;
  GtkLabel *label_tolerance;
  GtkSpinButton *spin_tolerance;
  GtkLabel *label_bests;
  GtkSpinButton *spin_bests;
  GtkLabel *label_population;
  GtkSpinButton *spin_population;
  GtkLabel *label_generations;
  GtkSpinButton *spin_generations;
  GtkLabel *label_mutation;
  GtkSpinButton *spin_mutation;
  GtkLabel *label_reproduction;
  GtkSpinButton *spin_reproduction;
  GtkLabel *label_adaptation;
  GtkSpinButton *spin_adaptation;
  GtkCheckButton *check_direction;
  GtkGrid *grid_direction;
  GtkRadioButton *button_direction[NDIRECTIONS];
  GtkLabel *label_steps;
  GtkSpinButton *spin_steps;
  GtkLabel *label_estimates;
  GtkSpinButton *spin_estimates;
  GtkLabel *label_relaxation;
  GtkSpinButton *spin_relaxation;
  GtkLabel *label_thresold;
  GtkSpinButton *spin_thresold;
  GtkScrolledWindow *scrolled_thresold;
  GtkFrame *frame_variable;
  GtkGrid *grid_variable;
  GtkComboBoxText *combo_variable;
  GtkButton *button_add_variable;
  GtkButton *button_remove_variable;
  GtkLabel *label_variable;
  GtkEntry *entry_variable;
  GtkLabel *label_min;
  GtkSpinButton *spin_min;
  GtkScrolledWindow *scrolled_min;
  GtkLabel *label_max;
  GtkSpinButton *spin_max;
  GtkScrolledWindow *scrolled_max;
  GtkCheckButton *check_minabs;
  GtkSpinButton *spin_minabs;
  GtkScrolledWindow *scrolled_minabs;
  GtkCheckButton *check_maxabs;
  GtkSpinButton *spin_maxabs;
  GtkScrolledWindow *scrolled_maxabs;
  GtkLabel *label_precision;
  GtkSpinButton *spin_precision;
  GtkLabel *label_sweeps;
  GtkSpinButton *spin_sweeps;
  GtkLabel *label_bits;
  GtkSpinButton *spin_bits;
  GtkLabel *label_step;
  GtkSpinButton *spin_step;
  GtkScrolledWindow *scrolled_step;
  GtkFrame *frame_experiment;
  GtkGrid *grid_experiment;
  GtkComboBoxText *combo_experiment;
  GtkButton *button_add_experiment;
  GtkButton *button_remove_experiment;
  GtkLabel *label_experiment;
  GtkFileChooserButton *button_experiment;
  GtkLabel *label_weight;
  GtkSpinButton *spin_weight;
  GtkCheckButton *check_template[MAX_NINPUTS];
  GtkFileChooserButton *button_template[MAX_NINPUTS];
  GdkPixbuf *logo;
  Experiment *experiment;
  Variable *variable;
  char *application_directory;
  gulong id_experiment;
  gulong id_experiment_name;
  gulong id_variable;
  gulong id_variable_label;
  gulong id_template[MAX_NINPUTS];
  gulong id_input[MAX_NINPUTS];
  unsigned int nexperiments;
  unsigned int nvariables;
} Window;

// Global variables
extern const char *logo[];
extern Options options[1];
extern Running running[1];
extern Window window[1];

// Public functions
unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
void input_save (char *filename);
void options_new ();
void running_new ();
unsigned int window_get_algorithm ();
unsigned int window_get_direction ();
unsigned int window_get_norm ();
void window_save_direction ();
int window_save ();
void window_run ();
void window_help ();
void window_update_direction ();
void window_update ();
void window_set_algorithm ();
void window_set_experiment ();
void window_remove_experiment ();
void window_add_experiment ();
void window_name_experiment ();
void window_weight_experiment ();
void window_inputs_experiment ();
void window_template_experiment (void *data);
void window_set_variable ();
void window_remove_variable ();
void window_add_variable ();
void window_label_variable ();
void window_precision_variable ();
void window_rangemin_variable ();
void window_rangemax_variable ();
void window_rangeminabs_variable ();
void window_rangemaxabs_variable ();
void window_update_variable ();
int window_read (char *filename);
void window_open ();
void window_new ();

#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.

```
{
#if HAVE_GTK
  char *buffer;
#endif

  // Starting pseudo-random numbers generator
  optimize->rng = gsl_rng_alloc (gsl_rng_taus2);

  // Allowing spaces in the XML data file
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
  printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

  // Resetting result and variables file names
  input->result = input->variables = NULL;

#if HAVE_GTK

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_direction = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory, LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  gtk_init (&argn, &argc);

  // Opening the main window
  window_new ();
  gtk_main ();

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_free (window->application_directory);

#else

  // Checking syntax
  if (argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_direction = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;
  if (argn > 2 && !strcmp (argc[1], "-nthreads"))
    {
      nthreads_direction = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-seed"))
        {
          optimize->seed = atoi (argc[2]);
          argc += 2;
          argn -= 2;
        }
    }
  else if (argn > 2 && !strcmp (argc[1], "-seed"))
    {
      optimize->seed = atoi (argc[2]);
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-nthreads"))
        {
          nthreads_direction = nthreads = atoi (argc[2]);
          if (!nthreads)
            {
              printf ("Bad threads number\n");
              return 2;
            }
          argc += 2;
          argn -= 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);
  printf ("seed=%lu\n", optimize->seed);

  // Checking arguments
  if (argn > 4 || argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }
  if (argn > 2)
    input->result = argc[2];
  if (argn == 4)
    input->variables = argc[3];

  // Making optimization
  if (input_open (argc[1]))
    optimize_open ();

  // Freeing memory
  optimize_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (optimize->rng);

  // Closing
  return 0;
}
```

Here is the call graph for this function:

5.8 main.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#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>
#ifdef G_OS_WIN32
#include <windows.h>
#elif !defined (BSD)
#include <alloca.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#if HAVE_GTK
#include <gio/gio.h>
#include <gtk/gtk.h>
#endif
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"
#if HAVE_GTK
#include "interface.h"
#endif

#define DEBUG 0


int
main (int argn, char **argc)
{
#if HAVE_GTK
  char *buffer;
#endif

  // Starting pseudo-random numbers generator
  optimize->rng = gsl_rng_alloc (gsl_rng_taus2);

  // Allowing spaces in the XML data file
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
  printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

  // Resetting result and variables file names
  input->result = input->variables = NULL;

#if HAVE_GTK

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_direction = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  gtk_init (&argn, &argc);

  // Opening the main window
  window_new ();
  gtk_main ();

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_free (window->application_directory);

#else

  // Checking syntax
  if (argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_direction = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;
  if (argn > 2 && !strcmp (argc[1], "-nthreads"))
    {
      nthreads_direction = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-seed"))
        {
          optimize->seed = atoi (argc[2]);
          argc += 2;
          argn -= 2;
        }
    }
  else if (argn > 2 && !strcmp (argc[1], "-seed"))
    {
      optimize->seed = atoi (argc[2]);
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-nthreads"))
        {
          nthreads_direction = nthreads = atoi (argc[2]);
          if (!nthreads)
            {
              printf ("Bad threads number\n");
              return 2;
            }
          argc += 2;
          argn -= 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);
  printf ("seed=%lu\n", optimize->seed);

  // Checking arguments
  if (argn > 4 || argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }
  if (argn > 2)
    input->result = argc[2];
  if (argn == 4)
    input->variables = argc[3];

  // Making optimization
  if (input_open (argc[1]))
    optimize_open ();

  // Freeing memory
  optimize_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (optimize->rng);

  // Closing
  return 0;
}
```

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

Copyright 2012-2016, all rights reserved.

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.

```
{
  char buffer2[64];
  char *buffert[MAX_NINPUTS] =
    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
  xmlDoc *doc;
  xmlNode *node, *child;
  xmlChar *buffer;
  char *msg;
  int error_code;
  unsigned int i;

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_OPTIMIZE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result = (char *) xmlGetProp (node, XML_RESULT);
      if (!input->result)
        input->result = (char *) xmlStrdup (result_name);
    }
  if (!input->variables)
    {
      input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
      if (!input->variables)
        input->variables = (char *) xmlStrdup (variables_name);
    }

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
```

```
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->
nsimulations;
      i += input->adaptation_ratio * input->
nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, XML_NBEST, 1, &error_code);
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node, XML_TOLERANCE, 0.,
                                           &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting direction search method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node, XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_DIRECTION);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->direction = DIRECTION_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->direction = DIRECTION_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the direction search");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the thresold
  input->thresold = xml_node_get_float_with_default (node, XML_THRESOLD, 0.,
                                                     &error_code);
  if (error_code)
    {
      msg = gettext ("Invalid thresold");
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child,
XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No optimization experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables], input->nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
          input->step[input->nvariables]
            = xml_node_get_float (child, XML_STEP, &error_code);
          if (error_code || input->step[input->nvariables] < 0.)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("bad step size"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      input->label = g_realloc
        (input->label, (1 + input->nvariables) * sizeof (char *));
      input->label[input->nvariables] = (char *) buffer;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      msg = gettext ("No optimization variables");
      goto exit_on_error;
    }
  buffer = NULL;

  // Obtaining the error norm
  if (xmlHasProp (node, XML_NORM))
    {
      buffer = xmlGetProp (node, XML_NORM);
      if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!xmlStrcmp (buffer, XML_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!xmlStrcmp (buffer, XML_P))
        {
          input->norm = ERROR_NORM_P;
          input->p = xml_node_get_float (node, XML_P, &error_code);
          if (!error_code)
            {
              msg = gettext ("Bad P parameter");
              goto exit_on_error;
            }
        }
      else if (!xmlStrcmp (buffer, XML_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          msg = gettext ("Unknown error norm");
          goto exit_on_error;
        }
      xmlFree (buffer);
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
  show_error (msg);
  input_free ();
#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}
```

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.

```
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->simulation_best[i - 1];
              optimize->error_best[i] = optimize->error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "optimize_best: end\n");
#endif
}
```

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.

```
{
#if DEBUG
  fprintf (stderr, "optimize_best_direction: start\n");
  fprintf (stderr,
           "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "optimize_best_direction: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_best_direction: end\n");
#endif
}
```

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.

```
{
  unsigned int i, j;
  double e;
#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.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
#endif
  x = optimize->direction[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG
  fprintf (stderr,
           "optimize_estimate_direction_coordinates: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
#endif
  return x;
}
```

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.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: start\n");
#endif
  x = optimize->direction[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
step[variable];
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_random: end\n");
#endif
  return x;
}
```

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         {
```

```
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "optimize_merge: end\n");
#endif
}
```

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.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}
```

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.

```
{
  double e;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}
```

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.

```
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG
  fprintf (stderr, "optimize_parse: start\n");
  fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
           experiment);
#endif

  // Opening input files
  for (i = 0; i < optimize->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG
      fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      optimize_input (simulation, &input[i][0], optimize->
file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG
  fprintf (stderr, "optimize_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (optimize->simulator);
  buffer3 = g_path_get_basename (optimize->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
            input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG
  fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (optimize->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (optimize->evaluator);
      buffer3 = g_path_get_basename (optimize->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, optimize->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG
      fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}
```

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.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}
```

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.

```
{
  GThread *thread[nthreads_direction];
  ParallelData data[nthreads_direction];
  unsigned int i, j, k, b;
#if DEBUG
  fprintf (stderr, "optimize_step_direction: start\n");
#endif
  for (i = 0; i < optimize->nestimates; ++i)
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG
      fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_direction (j, i);
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_direction == 1)
    optimize_direction_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_direction; ++i)
        {
          optimize->thread_direction[i]
            = simulation + optimize->nstart_direction
            + i * (optimize->nend_direction - optimize->nstart_direction)
            / nthreads_direction;
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: i=%u thread_direction=%u\n",
                   i, optimize->thread_direction[i]);
#endif
        }
      for (i = 0; i < nthreads_direction; ++i)
```

```
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) optimize_direction_thread, &data[i]);
        }
      for (i = 0; i < nthreads_direction; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "optimize_step_direction: end\n");
#endif
}
```

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.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

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

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#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>
#ifdef G_OS_WIN32
#include <windows.h>
#elif !defined (BSD)
#include <alloca.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#include "genetic/genetic.h"
#include "utils.h"
#include "optimize.h"

#define DEBUG 0


#ifdef G_OS_WIN32
#define RM "del"
#else
#define RM "rm"
#endif

int ntasks;
unsigned int nthreads;
unsigned int nthreads_direction;
GMutex mutex[1];
void (*optimize_algorithm) ();
double (*optimize_estimate_direction) (unsigned int variable,
                                       unsigned int estimate);
double (*optimize_norm) (unsigned int simulation);
Input input[1];
Optimize optimize[1];

const xmlChar *result_name = (xmlChar *) "result";
const xmlChar *variables_name = (xmlChar *) "variables";

const xmlChar *template[MAX_NINPUTS] = {
  XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3, XML_TEMPLATE4,
  XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7, XML_TEMPLATE8
};


const char *format[NPRECISIONS] = {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
};

const double precision[NPRECISIONS] = {
  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
};

void
input_new ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "input_new: start\n");
#endif
  input->nvariables = input->nexperiments = input->ninputs = input->nsteps = 0;
  input->simulator = input->evaluator = input->directory = input->name = NULL;
  input->experiment = input->label = NULL;
  input->precision = input->nsweeps = input->nbits = NULL;
  input->rangemin = input->rangemax = input->rangeminabs = input->rangemaxabs
    = input->weight = input->step = NULL;
  for (i = 0; i < MAX_NINPUTS; ++i)
    input->template[i] = NULL;
#if DEBUG
  fprintf (stderr, "input_new: end\n");
#endif
}

void
input_free ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "input_free: start\n");
#endif
```

```
  g_free (input->name);
  g_free (input->directory);
  for (i = 0; i < input->nexperiments; ++i)
    {
      xmlFree (input->experiment[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlFree (input->template[j][i]);
      g_free (input->template[j]);
    }
  g_free (input->experiment);
  for (i = 0; i < input->ninputs; ++i)
    g_free (input->template[i]);
  for (i = 0; i < input->nvariables; ++i)
    xmlFree (input->label[i]);
  g_free (input->label);
  g_free (input->precision);
  g_free (input->rangemin);
  g_free (input->rangemax);
  g_free (input->rangeminabs);
  g_free (input->rangemaxabs);
  g_free (input->weight);
  g_free (input->step);
  g_free (input->nsweeps);
  g_free (input->nbits);
  xmlFree (input->evaluator);
  xmlFree (input->simulator);
  xmlFree (input->result);
  xmlFree (input->variables);
  input->nexperiments = input->ninputs = input->nvariables = input->nsteps = 0;
#if DEBUG
  fprintf (stderr, "input_free: end\n");
#endif
}

int
input_open (char *filename)
{
  char buffer2[64];
  char *buffert[MAX_NINPUTS] =
    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
  xmlDoc *doc;
  xmlNode *node, *child;
  xmlChar *buffer;
  char *msg;
  int error_code;
  unsigned int i;

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_OPTIMIZE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result = (char *) xmlGetProp (node, XML_RESULT);
      if (!input->result)
        input->result = (char *) xmlStrdup (result_name);
    }
  if (!input->variables)
    {
      input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
      if (!input->variables)
        input->variables = (char *) xmlStrdup (variables_name);
    }

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
```

```
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node, XML_TOLERANCE, 0.,
                                           &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting direction search method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node, XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_DIRECTION);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->direction = DIRECTION_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->direction = DIRECTION_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the direction search");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the thresold
  input->thresold = xml_node_get_float_with_default (node, XML_THRESOLD, 0.,
                                                     &error_code);
  if (error_code)
    {
      msg = gettext ("Invalid thresold");
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child,
XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No optimization experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
```

```
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables], input->nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
          input->step[input->nvariables]
            = xml_node_get_float (child, XML_STEP, &error_code);
          if (error_code || input->step[input->nvariables] < 0.)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("bad step size"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      input->label = g_realloc
        (input->label, (1 + input->nvariables) * sizeof (char *));
      input->label[input->nvariables] = (char *) buffer;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      msg = gettext ("No optimization variables");
      goto exit_on_error;
    }
  buffer = NULL;

  // Obtaining the error norm
  if (xmlHasProp (node, XML_NORM))
    {
      buffer = xmlGetProp (node, XML_NORM);
      if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!xmlStrcmp (buffer, XML_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!xmlStrcmp (buffer, XML_P))
        {
          input->norm = ERROR_NORM_P;
          input->p = xml_node_get_float (node, XML_P, &error_code);
          if (!error_code)
            {
              msg = gettext ("Bad P parameter");
              goto exit_on_error;
            }
        }
      else if (!xmlStrcmp (buffer, XML_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          msg = gettext ("Unknown error norm");
          goto exit_on_error;
        }
      xmlFree (buffer);
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
  show_error (msg);
  input_free ();
#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}

void
optimize_input (unsigned int simulation, char *input, GMappedFile * template)
{
  unsigned int i;
  char buffer[32], value[32], *buffer2, *buffer3, *content;
  FILE *file;
  gsize length;
  GRegex *regex;

#if DEBUG
  fprintf (stderr, "optimize_input: start\n");
#endif

  // Checking the file
  if (!template)
    goto optimize_input_end;

  // Opening template
  content = g_mapped_file_get_contents (template);
  length = g_mapped_file_get_length (template);
#if DEBUG
  fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
#endif
  file = g_fopen (input, "w");

  // Parsing template
  for (i = 0; i < optimize->nvariables; ++i)
    {
#if DEBUG
      fprintf (stderr, "optimize_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             optimize->label[i], 0, NULL);
#if DEBUG
          fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             optimize->label[i], 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      snprintf (value, 32, format[optimize->precision[i]],
                optimize->value[simulation * optimize->nvariables + i]);
```

```c

#if DEBUG
      fprintf (stderr, "optimize_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

optimize_input_end:
#if DEBUG
  fprintf (stderr, "optimize_input: end\n");
#endif
  return;
}

double
optimize_parse (unsigned int simulation, unsigned int experiment)
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG
  fprintf (stderr, "optimize_parse: start\n");
  fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
           experiment);
#endif

  // Opening input files
  for (i = 0; i < optimize->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG
      fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG
  fprintf (stderr, "optimize_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (optimize->simulator);
  buffer3 = g_path_get_basename (optimize->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
            input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG
  fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (optimize->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (optimize->evaluator);
      buffer3 = g_path_get_basename (optimize->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, optimize->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG
      fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}

double
optimize_norm_euclidian (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}

double
optimize_norm_maximum (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_maximum: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e = fmax (e, ei);
    }
#if DEBUG
  fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_maximum: end\n");
#endif
  return e;
}

double
optimize_norm_p (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_p: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e += pow (ei, optimize->p);
    }
  e = pow (e, 1. / optimize->p);
#if DEBUG
  fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_p: end\n");
#endif
  return e;
}

double
optimize_norm_taxicab (unsigned int simulation)
{
  double e;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}

void
optimize_print ()
{
  unsigned int i;
  char buffer[512];
#if HAVE_MPI
  if (optimize->mpi_rank)
    return;
#endif
  printf ("%s\n", gettext ("Best result"));
  fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
  printf ("error = %.15le\n", optimize->error_old[0]);
  fprintf (optimize->file_result, "error = %.15le\n", optimize->error_old[0]);
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 512, "%s = %s\n",
                optimize->label[i], format[optimize->precision[i]]);
      printf (buffer, optimize->value_old[i]);
      fprintf (optimize->file_result, buffer, optimize->value_old[i]);
    }
  fflush (optimize->file_result);
}

void
optimize_save_variables (unsigned int simulation, double error)
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}

void
optimize_best (unsigned int simulation, double value)
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->simulation_best[i - 1];
              optimize->error_best[i] = optimize->error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "optimize_best: end\n");
#endif
}

void
optimize_sequential ()
{
  unsigned int i;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_sequential: start\n");
  fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
           optimize->nstart, optimize->nend);
#endif
  for (i = optimize->nstart; i < optimize->nend; ++i)
    {
      e = optimize_norm (i);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG
      fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_sequential: end\n");
#endif
}

void *
optimize_thread (ParallelData * data)
{
  unsigned int i, thread;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

void
```

```
optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
                double *error_best)
{
  unsigned int i, j, k, s[optimize->nbest];
  double e[optimize->nbest];
#if DEBUG
  fprintf (stderr, "optimize_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == optimize->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "optimize_merge: end\n");
#endif
}

#if HAVE_MPI
void
optimize_synchronise ()
{
  unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
  double error_best[optimize->nbest];
  MPI_Status mpi_stat;
#if DEBUG
  fprintf (stderr, "optimize_synchronise: start\n");
#endif
  if (optimize->mpi_rank == 0)
    {
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          optimize_merge (nsaveds, simulation_best, error_best);
          MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
          if (stop)
            optimize->stop = 1;
        }
      for (i = 1; i < ntasks; ++i)
        MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
    }
  else
    {
      MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
      MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
      if (stop)
        optimize->stop = 1;
    }
#if DEBUG
  fprintf (stderr, "optimize_synchronise: end\n");
#endif
}
#endif

void
optimize_sweep ()
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG
  fprintf (stderr, "optimize_sweep: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += l * (optimize->rangemax[j] - optimize->rangemin[j])
              / (optimize->nsweeps[j] - 1);
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG
  fprintf (stderr, "optimize_sweep: end\n");
#endif
}

void
optimize_MonteCarlo ()
{
  unsigned int i, j;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG
  fprintf (stderr, "optimize_MonteCarlo: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    for (j = 0; j < optimize->nvariables; ++j)
      optimize->value[i * optimize->nvariables + j]
        = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
        * (optimize->rangemax[j] - optimize->rangemin[j]);
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
```

```
#endif
#if DEBUG
  fprintf (stderr, "optimize_MonteCarlo: end\n");
#endif
}

void
optimize_best_direction (unsigned int simulation, double value)
{
#if DEBUG
  fprintf (stderr, "optimize_best_direction: start\n");
  fprintf (stderr,
           "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "optimize_best_direction: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_best_direction: end\n");
#endif
}

void
optimize_direction_sequential (unsigned int simulation)
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_direction_sequential: start\n");
  fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
           "nend_direction=%u\n",
           optimize->nstart_direction, optimize->nend_direction);
#endif
  for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
    {
      j = simulation + i;
      e = optimize_norm (j);
      optimize_best_direction (j, e);
      optimize_save_variables (j, e);
      if (e < optimize->thresold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG
      fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_direction_sequential: end\n");
#endif
}

void *
optimize_direction_thread (ParallelData * data)
{
  unsigned int i, thread;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
           thread,
           optimize->thread_direction[thread],
           optimize->thread_direction[thread + 1]);
#endif
  for (i = optimize->thread_direction[thread];
       i < optimize->thread_direction[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best_direction (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
```

```
        break;
#if DEBUG
      fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

double
optimize_estimate_direction_random (unsigned int variable,
                                    unsigned int estimate)
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: start\n");
#endif
  x = optimize->direction[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_random: end\n");
#endif
  return x;
}

double
optimize_estimate_direction_coordinates (unsigned int variable,
                                         unsigned int estimate)
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
#endif
  x = optimize->direction[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG
  fprintf (stderr,
           "optimize_estimate_direction_coordinates: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
#endif
  return x;
}

void
optimize_step_direction (unsigned int simulation)
{
  GThread *thread[nthreads_direction];
  ParallelData data[nthreads_direction];
  unsigned int i, j, k, b;
#if DEBUG
  fprintf (stderr, "optimize_step_direction: start\n");
#endif
  for (i = 0; i < optimize->nestimates; ++i)
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG
      fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_direction (j, i);
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_direction == 1)
    optimize_direction_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_direction; ++i)
        {
          optimize->thread_direction[i]
            = simulation + optimize->nstart_direction
            + i * (optimize->nend_direction - optimize->nstart_direction)
            / nthreads_direction;
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: i=%u thread_direction=%u\n",
                   i, optimize->thread_direction[i]);
#endif
        }
      for (i = 0; i < nthreads_direction; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) optimize_direction_thread, &data[i]);
        }
      for (i = 0; i < nthreads_direction; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "optimize_step_direction: end\n");
#endif
}

void
optimize_direction ()
{
  unsigned int i, j, k, b, s, adjust;
#if DEBUG
  fprintf (stderr, "optimize_direction: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    optimize->direction[i] = 0.;
  b = optimize->simulation_best[0] * optimize->nvariables;
  s = optimize->nsimulations;
  adjust = 1;
  for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
    {
#if DEBUG
      fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      optimize_step_direction (s);
      k = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG
      fprintf (stderr, "optimize_direction: step=%u best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      if (k == b)
        {
          if (adjust)
            for (j = 0; j < optimize->nvariables; ++j)
              optimize->step[j] *= 0.5;
          for (j = 0; j < optimize->nvariables; ++j)
            optimize->direction[j] = 0.;
          adjust = 1;
        }
      else
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
#if DEBUG
              fprintf (stderr,
                       "optimize_direction: best%u=%.14le old%u=%.14le\n",
                       j, optimize->value[k + j], j, optimize->value[b + j]);
#endif
              optimize->direction[j]
                = (1. - optimize->relaxation) * optimize->direction[j]
                + optimize->relaxation
                * (optimize->value[k + j] - optimize->value[b + j]);
#if DEBUG
              fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
                       j, optimize->direction[j]);
#endif
            }
          adjust = 0;
        }
    }
#if DEBUG
  fprintf (stderr, "optimize_direction: end\n");
#endif
}

double
optimize_genetic_objective (Entity * entity)
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "optimize_genetic_objective: start\n");
#endif
  for (j = 0; j < optimize->nvariables; ++j)
    {
      optimize->value[entity->id * optimize->nvariables + j]
        = genetic_get_variable (entity, optimize->genetic_variable + j);
    }
  objective = optimize_norm (entity->id);
  g_mutex_lock (mutex);
  for (j = 0; j < optimize->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
      fprintf (optimize->file_variables, buffer,
               genetic_get_variable (entity, optimize->genetic_variable + j));
    }
  fprintf (optimize->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG
  fprintf (stderr, "optimize_genetic_objective: end\n");
#endif
  return objective;
}

void
optimize_genetic ()
{
  char *best_genome;
  double best_objective, *best_variable;
#if DEBUG
  fprintf (stderr, "optimize_genetic: start\n");
  fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
           nthreads);
  fprintf (stderr,
           "optimize_genetic: nvariables=%u population=%u generations=%u\n",
           optimize->nvariables, optimize->nsimulations, optimize->niterations);
  fprintf (stderr,
           "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
           optimize->mutation_ratio, optimize->reproduction_ratio,
           optimize->adaptation_ratio);
#endif
  genetic_algorithm_default (optimize->nvariables,
                             optimize->genetic_variable,
                             optimize->nsimulations,
                             optimize->niterations,
                             optimize->mutation_ratio,
                             optimize->reproduction_ratio,
                             optimize->adaptation_ratio,
                             optimize->thresold,
                             &optimize_genetic_objective,
                             &best_genome, &best_variable, &best_objective);
#if DEBUG
  fprintf (stderr, "optimize_genetic: the best\n");
#endif
  optimize->error_old = (double *) g_malloc (sizeof (double));
  optimize->value_old
    = (double *) g_malloc (optimize->nvariables * sizeof (double));
  optimize->error_old[0] = best_objective;
  memcpy (optimize->value_old, best_variable,
          optimize->nvariables * sizeof (double));
  g_free (best_genome);
  g_free (best_variable);
  optimize_print ();
#if DEBUG
  fprintf (stderr, "optimize_genetic: end\n");
#endif
}

void
optimize_save_old ()
{
  unsigned int i, j;
#if DEBUG
```

```
  fprintf (stderr, "optimize_save_old: start\n");
  fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
#endif
  memcpy (optimize->error_old, optimize->error_best,
          optimize->nbest * sizeof (double));
  for (i = 0; i < optimize->nbest; ++i)
    {
      j = optimize->simulation_best[i];
#if DEBUG
      fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
#endif
      memcpy (optimize->value_old + i * optimize->nvariables,
              optimize->value + j * optimize->nvariables,
              optimize->nvariables * sizeof (double));
    }
#if DEBUG
  for (i = 0; i < optimize->nvariables; ++i)
    fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
             i, optimize->value_old[i]);
  fprintf (stderr, "optimize_save_old: end\n");
#endif
}

void
optimize_merge_old ()
{
  unsigned int i, j, k;
  double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
    *enew, *eold;
#if DEBUG
  fprintf (stderr, "optimize_merge_old: start\n");
#endif
  enew = optimize->error_best;
  eold = optimize->error_old;
  i = j = k = 0;
  do
    {
      if (*enew < *eold)
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value
                  + optimize->simulation_best[i] * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *enew;
          ++k;
          ++enew;
          ++i;
        }
      else
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value_old + j * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *eold;
          ++k;
          ++eold;
          ++j;
        }
    }
  while (k < optimize->nbest);
  memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
  memcpy (optimize->error_old, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "optimize_merge_old: end\n");
#endif
}

void
optimize_refine ()
{
  unsigned int i, j;
  double d;
#if HAVE_MPI
  MPI_Status mpi_stat;
#endif
#if DEBUG
  fprintf (stderr, "optimize_refine: start\n");
#endif
#if HAVE_MPI
  if (!optimize->mpi_rank)
    {
#endif
      for (j = 0; j < optimize->nvariables; ++j)
        {
          optimize->rangemin[j] = optimize->rangemax[j]
```

```
            = optimize->value_old[j];
        }
      for (i = 0; ++i < optimize->nbest;)
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
              optimize->rangemin[j]
                = fmin (optimize->rangemin[j],
                        optimize->value_old[i * optimize->nvariables + j]);
              optimize->rangemax[j]
                = fmax (optimize->rangemax[j],
                        optimize->value_old[i * optimize->nvariables + j]);
            }
        }
      for (j = 0; j < optimize->nvariables; ++j)
        {
          d = optimize->tolerance
            * (optimize->rangemax[j] - optimize->rangemin[j]);
          switch (optimize->algorithm)
            {
            case ALGORITHM_MONTE_CARLO:
              d *= 0.5;
              break;
            default:
              if (optimize->nsweeps[j] > 1)
                d /= optimize->nsweeps[j] - 1;
              else
                d = 0.;
            }
          optimize->rangemin[j] -= d;
          optimize->rangemin[j]
            = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
          optimize->rangemax[j] += d;
          optimize->rangemax[j]
            = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
          printf ("%s min=%lg max=%lg\n", optimize->label[j],
                  optimize->rangemin[j], optimize->rangemax[j]);
          fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
                   optimize->label[j], optimize->rangemin[j],
                   optimize->rangemax[j]);
        }
#if HAVE_MPI
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
          MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
        }
    }
  else
    {
      MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD, &mpi_stat);
      MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD, &mpi_stat);
    }
#endif
#if DEBUG
  fprintf (stderr, "optimize_refine: end\n");
#endif
}

void
optimize_step ()
{
#if DEBUG
  fprintf (stderr, "optimize_step: start\n");
#endif
  optimize_algorithm ();
  if (optimize->nsteps)
    optimize_direction ();
#if DEBUG
  fprintf (stderr, "optimize_step: end\n");
#endif
}

void
optimize_iterate ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_iterate: start\n");
#endif
  optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
  optimize->value_old = (double *)
    g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
  optimize_step ();
  optimize_save_old ();
  optimize_refine ();
  optimize_print ();
  for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
    {
      optimize_step ();
      optimize_merge_old ();
      optimize_refine ();
      optimize_print ();
    }
#if DEBUG
  fprintf (stderr, "optimize_iterate: end\n");
#endif
}

void
optimize_free ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "optimize_free: start\n");
#endif
  for (j = 0; j < optimize->ninputs; ++j)
    {
      for (i = 0; i < optimize->nexperiments; ++i)
        g_mapped_file_unref (optimize->file[j][i]);
      g_free (optimize->file[j]);
    }
  g_free (optimize->error_old);
  g_free (optimize->value_old);
  g_free (optimize->value);
  g_free (optimize->genetic_variable);
  g_free (optimize->rangemax);
  g_free (optimize->rangemin);
#if DEBUG
  fprintf (stderr, "optimize_free: end\n");
#endif
}

void
optimize_open ()
{
  GTimeZone *tz;
  GDateTime *t0, *t;
  unsigned int i, j, *nbits;

#if DEBUG
  char *buffer;
  fprintf (stderr, "optimize_open: start\n");
#endif

  // Getting initial time
#if DEBUG
  fprintf (stderr, "optimize_open: getting initial time\n");
#endif
  tz = g_time_zone_new_utc ();
  t0 = g_date_time_new_now (tz);

  // Obtaining and initing the pseudo-random numbers generator seed
#if DEBUG
  fprintf (stderr, "optimize_open: getting initial seed\n");
#endif
  optimize->seed = input->seed;
  gsl_rng_set (optimize->rng, optimize->seed);

  // Replacing the working directory
#if DEBUG
  fprintf (stderr, "optimize_open: replacing the working directory\n");
#endif
  g_chdir (input->directory);

  // Getting results file names
  optimize->result = input->result;
  optimize->variables = input->variables;

  // Obtaining the simulator file
  optimize->simulator = input->simulator;

  // Obtaining the evaluator file
  optimize->evaluator = input->evaluator;

  // Reading the algorithm
  optimize->algorithm = input->algorithm;
  switch (optimize->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      optimize_algorithm = optimize_MonteCarlo;
      break;
    case ALGORITHM_SWEEP:
      optimize_algorithm = optimize_sweep;
      break;
    default:
      optimize_algorithm = optimize_genetic;
      optimize->mutation_ratio = input->mutation_ratio;
      optimize->reproduction_ratio = input->reproduction_ratio;
      optimize->adaptation_ratio = input->adaptation_ratio;
    }
  optimize->nvariables = input->nvariables;
  optimize->nsimulations = input->nsimulations;
  optimize->niterations = input->niterations;
  optimize->nbest = input->nbest;
  optimize->tolerance = input->tolerance;
  optimize->nsteps = input->nsteps;
  optimize->nestimates = 0;
  optimize->thresold = input->thresold;
  optimize->stop = 0;
  if (input->nsteps)
    {
      optimize->relaxation = input->relaxation;
      switch (input->direction)
        {
        case DIRECTION_METHOD_COORDINATES:
          optimize->nestimates = 2 * optimize->nvariables;
          optimize_estimate_direction =
optimize_estimate_direction_coordinates;
          break;
        default:
          optimize->nestimates = input->nestimates;
          optimize_estimate_direction =
optimize_estimate_direction_random;
        }
    }

#if DEBUG
  fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
#endif
  optimize->simulation_best
    = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
  optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));

  // Reading the experimental data
#if DEBUG
  buffer = g_get_current_dir ();
  fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
  g_free (buffer);
#endif
  optimize->nexperiments = input->nexperiments;
  optimize->ninputs = input->ninputs;
  optimize->experiment = input->experiment;
  optimize->weight = input->weight;
  for (i = 0; i < input->ninputs; ++i)
    {
      optimize->template[i] = input->template[i];
      optimize->file[i]
        = g_malloc (input->nexperiments * sizeof (GMappedFile *));
    }
  for (i = 0; i < input->nexperiments; ++i)
    {
#if DEBUG
      fprintf (stderr, "optimize_open: i=%u\n", i);
      fprintf (stderr, "optimize_open: experiment=%s\n",
               optimize->experiment[i]);
      fprintf (stderr, "optimize_open: weight=%lg\n", optimize->weight[i]);
#endif
      for (j = 0; j < input->ninputs; ++j)
        {
#if DEBUG
          fprintf (stderr, "optimize_open: template%u\n", j + 1);
          fprintf (stderr, "optimize_open: experiment=%u template%u=%s\n",
                   i, j + 1, optimize->template[j][i]);
#endif
          optimize->file[j][i]
            = g_mapped_file_new (input->template[j][i], 0, NULL);
        }
    }

  // Reading the variables data
#if DEBUG
  fprintf (stderr, "optimize_open: reading variables\n");
#endif
  optimize->label = input->label;
  j = input->nvariables * sizeof (double);
  optimize->rangemin = (double *) g_malloc (j);
  optimize->rangemax = (double *) g_malloc (j);
  memcpy (optimize->rangemin, input->rangemin, j);
  memcpy (optimize->rangemax, input->rangemax, j);
  optimize->rangeminabs = input->rangeminabs;
  optimize->rangemaxabs = input->rangemaxabs;
  optimize->precision = input->precision;
  optimize->nsweeps = input->nsweeps;
  optimize->step = input->step;
  nbits = input->nbits;
  if (input->algorithm == ALGORITHM_SWEEP)
    {
      optimize->nsimulations = 1;
      for (i = 0; i < input->nvariables; ++i)
        {
          if (input->algorithm == ALGORITHM_SWEEP)
            {
              optimize->nsimulations *= input->nsweeps[i];
#if DEBUG
              fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
                       optimize->nsweeps[i], optimize->nsimulations);
#endif
            }
        }
    }
  if (optimize->nsteps)
    optimize->direction
      = (double *) alloca (optimize->nvariables * sizeof (double));

  // Setting error norm
  switch (input->norm)
    {
    case ERROR_NORM_EUCLIDIAN:
      optimize_norm = optimize_norm_euclidian;
      break;
    case ERROR_NORM_MAXIMUM:
      optimize_norm = optimize_norm_maximum;
      break;
    case ERROR_NORM_P:
      optimize_norm = optimize_norm_p;
      optimize->p = input->p;
      break;
    default:
      optimize_norm = optimize_norm_taxicab;
    }

  // Allocating values
#if DEBUG
  fprintf (stderr, "optimize_open: allocating variables\n");
  fprintf (stderr, "optimize_open: nvariables=%u\n", optimize->nvariables);
#endif
  optimize->genetic_variable = NULL;
  if (optimize->algorithm == ALGORITHM_GENETIC)
    {
      optimize->genetic_variable = (GeneticVariable *)
        g_malloc (optimize->nvariables * sizeof (GeneticVariable));
      for (i = 0; i < optimize->nvariables; ++i)
        {
#if DEBUG
          fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
                   i, optimize->rangemin[i], optimize->rangemax[i], nbits[i]);
#endif
          optimize->genetic_variable[i].minimum = optimize->
rangemin[i];
          optimize->genetic_variable[i].maximum = optimize->
rangemax[i];
          optimize->genetic_variable[i].nbits = nbits[i];
        }
    }
#if DEBUG
  fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
           optimize->nvariables, optimize->nsimulations);
#endif
  optimize->value = (double *)
    g_malloc ((optimize->nsimulations
               + optimize->nestimates * optimize->nsteps)
              * optimize->nvariables * sizeof (double));

  // Calculating simulations to perform for each task
#if HAVE_MPI
#if DEBUG
  fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
           optimize->mpi_rank, ntasks);
#endif
  optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
ntasks;
  optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
ntasks;
```

```
  if (optimize->nsteps)
    {
      optimize->nstart_direction
        = optimize->mpi_rank * optimize->nestimates / ntasks;
      optimize->nend_direction
        = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
    }
#else
  optimize->nstart = 0;
  optimize->nend = optimize->nsimulations;
  if (optimize->nsteps)
    {
      optimize->nstart_direction = 0;
      optimize->nend_direction = optimize->nestimates;
    }
#endif
#if DEBUG
  fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
           optimize->nend);
#endif

  // Calculating simulations to perform for each thread
  optimize->thread
    = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
  for (i = 0; i <= nthreads; ++i)
    {
      optimize->thread[i] = optimize->nstart
        + i * (optimize->nend - optimize->nstart) / nthreads;
#if DEBUG
      fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
               optimize->thread[i]);
#endif
    }
  if (optimize->nsteps)
    optimize->thread_direction = (unsigned int *)
      alloca ((1 + nthreads_direction) * sizeof (unsigned int));

  // Opening result files
  optimize->file_result = g_fopen (optimize->result, "w");
  optimize->file_variables = g_fopen (optimize->variables, "w");

  // Performing the algorithm
  switch (optimize->algorithm)
    {
      // Genetic algorithm
    case ALGORITHM_GENETIC:
      optimize_genetic ();
      break;

      // Iterative algorithm
    default:
      optimize_iterate ();
    }

  // Getting calculation time
  t = g_date_time_new_now (tz);
  optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
  g_date_time_unref (t);
  g_date_time_unref (t0);
  g_time_zone_unref (tz);
  printf ("%s = %.6lg s\n",
          gettext ("Calculation time"), optimize->calculation_time);
  fprintf (optimize->file_result, "%s = %.6lg s\n",
           gettext ("Calculation time"), optimize->calculation_time);

  // Closing result files
  fclose (optimize->file_variables);
  fclose (optimize->file_result);

#if DEBUG
  fprintf (stderr, "optimize_open: end\n");
#endif
}
```

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: $\sqrt{\sum_i \left(w_i\,x_i\right)^2}$.
- ***ERROR_NORM_MAXIMUM*** Maximum norm: $\max_i \left|w_i\,x_i\right|$.
- ***ERROR_NORM_P*** P-norm $\sqrt[p]{\sum_i \left|w_i\,x_i\right|^p}$.
- ***ERROR_NORM_TAXICAB*** Taxicab norm $\sum_i \left|w_i\,x_i\right|$.

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;
```

```

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_OPTIMIZE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result = (char *) xmlGetProp (node, XML_RESULT);
      if (!input->result)
        input->result = (char *) xmlStrdup (result_name);
    }
  if (!input->variables)
    {
      input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
      if (!input->variables)
        input->variables = (char *) xmlStrdup (variables_name);
    }

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, XML_NBEST, 1, &error_code);
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node, XML_TOLERANCE, 0.,
                                           &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting direction search method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node, XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_DIRECTION);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->direction = DIRECTION_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->direction = DIRECTION_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the direction search");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the thresold
  input->thresold = xml_node_get_float_with_default (node, XML_THRESOLD, 0.,
                                                     &error_code);
  if (error_code)
    {
      msg = gettext ("Invalid thresold");
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child, XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No optimization experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->
nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables],
input->nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
```

```
     input->step[input->nvariables]
      = xml_node_get_float (child, XML_STEP, &error_code);
     if (error_code || input->step[input->nvariables] < 0.)
      {
       snprintf (buffer2, 64, "%s %s: %s",
            gettext ("Variable"),
            buffer, gettext ("bad step size"));
       msg = buffer2;
       goto exit_on_error;
      }
    }
   input->label = g_realloc
    (input->label, (1 + input->nvariables) * sizeof (char *));
   input->label[input->nvariables] = (char *) buffer;
   ++input->nvariables;
  }
 if (!input->nvariables)
  {
   msg = gettext ("No optimization variables");
   goto exit_on_error;
  }
 buffer = NULL;

 // Obtaining the error norm
 if (xmlHasProp (node, XML_NORM))
  {
   buffer = xmlGetProp (node, XML_NORM);
   if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
    input->norm = ERROR_NORM_EUCLIDIAN;
   else if (!xmlStrcmp (buffer, XML_MAXIMUM))
    input->norm = ERROR_NORM_MAXIMUM;
   else if (!xmlStrcmp (buffer, XML_P))
    {
     input->norm = ERROR_NORM_P;
     input->p = xml_node_get_float (node, XML_P, &error_code);
     if (!error_code)
      {
       msg = gettext ("Bad P parameter");
       goto exit_on_error;
      }
    }
   else if (!xmlStrcmp (buffer, XML_TAXICAB))
    input->norm = ERROR_NORM_TAXICAB;
   else
    {
     msg = gettext ("Unknown error norm");
     goto exit_on_error;
    }
   xmlFree (buffer);
  }
 else
  input->norm = ERROR_NORM_EUCLIDIAN;

 // Getting the working directory
 input->directory = g_path_get_dirname (filename);
 input->name = g_path_get_basename (filename);

 // Closing the XML document
 xmlFreeDoc (doc);

#if DEBUG
 fprintf (stderr, "input_open: end\n");
#endif
 return 1;

exit_on_error:
 xmlFree (buffer);
 xmlFreeDoc (doc);
 show_error (msg);
 input_free ();
#if DEBUG
 fprintf (stderr, "input_open: end\n");
#endif
 return 0;
}
```

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.

```
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->simulation_best[i - 1];
              optimize->error_best[i] = optimize->error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "optimize_best: end\n");
#endif
}
```

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.

```
{
#if DEBUG
  fprintf (stderr, "optimize_best_direction: start\n");
  fprintf (stderr,
           "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "optimize_best_direction: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_best_direction: end\n");
#endif
}
```

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.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
           thread,
           optimize->thread_direction[thread],
           optimize->thread_direction[thread + 1]);
#endif
  for (i = optimize->thread_direction[thread];
       i < optimize->thread_direction[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best_direction (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG
      fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_direction_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

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.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
#endif
  x = optimize->direction[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG
  fprintf (stderr,
           "optimize_estimate_direction_coordinates: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
#endif
  return x;
}
```

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.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: start\n");
#endif
  x = optimize->direction[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
#if DEBUG
  fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_direction_random: end\n");
#endif
  return x;
}
```

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.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "optimize_genetic_objective: start\n");
#endif
  for (j = 0; j < optimize->nvariables; ++j)
    {
      optimize->value[entity->id * optimize->nvariables + j]
        = genetic_get_variable (entity, optimize->genetic_variable + j);
    }
  objective = optimize_norm (entity->id);
  g_mutex_lock (mutex);
  for (j = 0; j < optimize->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
      fprintf (optimize->file_variables, buffer,
               genetic_get_variable (entity, optimize->genetic_variable + j));
    }
  fprintf (optimize->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG
  fprintf (stderr, "optimize_genetic_objective: end\n");
#endif
  return objective;
}
```

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.

```
{
  unsigned int i;
  char buffer[32], value[32], *buffer2, *buffer3, *content;
  FILE *file;
  gsize length;
  GRegex *regex;

#if DEBUG
  fprintf (stderr, "optimize_input: start\n");
#endif

  // Checking the file
  if (!template)
    goto optimize_input_end;

  // Opening template
  content = g_mapped_file_get_contents (template);
  length = g_mapped_file_get_length (template);
#if DEBUG
  fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
#endif
  file = g_fopen (input, "w");

  // Parsing template
  for (i = 0; i < optimize->nvariables; ++i)
    {
#if DEBUG
      fprintf (stderr, "optimize_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             optimize->label[i], 0, NULL);
#if DEBUG
          fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             optimize->label[i], 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      snprintf (value, 32, format[optimize->precision[i]],
                optimize->value[simulation * optimize->
nvariables + i]);

#if DEBUG
      fprintf (stderr, "optimize_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

optimize_input_end:
#if DEBUG
  fprintf (stderr, "optimize_input: end\n");
#endif
  return;
}
```

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.

```
{
  unsigned int i, j, k, s[optimize->nbest];
  double e[optimize->nbest];
#if DEBUG
  fprintf (stderr, "optimize_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == optimize->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "optimize_merge: end\n");
#endif
}
```

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.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}
```

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.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_maximum: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e = fmax (e, ei);
    }
#if DEBUG
  fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_maximum: end\n");
#endif
  return e;
}
```

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.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_p: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e += pow (ei, optimize->p);
    }
  e = pow (e, 1. / optimize->p);
#if DEBUG
  fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_p: end\n");
#endif
  return e;
}
```

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.

```
{
  double e;
  unsigned int i;
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}
```

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);
```

```
    }

  // Removing files
#if !DEBUG
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}
```

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.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->
nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}
```

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.

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

```
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->
nvariables;
#if DEBUG
      fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_direction (j,
i);
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_direction == 1)
    optimize_direction_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_direction; ++i)
        {
          optimize->thread_direction[i]
            = simulation + optimize->nstart_direction
            + i * (optimize->nend_direction - optimize->
nstart_direction)
            / nthreads_direction;
#if DEBUG
          fprintf (stderr,
                   "optimize_step_direction: i=%u thread_direction=%u\n",
                   i, optimize->thread_direction[i]);
#endif
        }
      for (i = 0; i < nthreads_direction; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) optimize_direction_thread, &data[i]);
        }
      for (i = 0; i < nthreads_direction; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "optimize_step_direction: end\n");
#endif
}
```

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.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->thresold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

Here is the call graph for this function:

5.12 optimize.h

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef OPTIMIZE__H
#define OPTIMIZE__H 1

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

enum DirectionMethod
{
  DIRECTION_METHOD_COORDINATES = 0,
  DIRECTION_METHOD_RANDOM = 1,
};

enum ErrorNorm
```

```
{
  ERROR_NORM_EUCLIDIAN = 0,
  ERROR_NORM_MAXIMUM = 1,
  ERROR_NORM_P = 2,
  ERROR_NORM_TAXICAB = 3
};

typedef struct
{
  char **template[MAX_NINPUTS];
  char **experiment;
  char **label;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  char *directory;
  char *name;
  double *rangemin;
  double *rangemax;
  double *rangeminabs;
  double *rangemaxabs;
  double *weight;
  double *step;
  unsigned int *precision;
  unsigned int *nsweeps;
  unsigned int *nbits;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  double p;
  double thresold;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int ninputs;
  unsigned int nsimulations;
  unsigned int algorithm;
  unsigned int nsteps;
  unsigned int direction;
  unsigned int nestimates;
  unsigned int niterations;
  unsigned int nbest;
  unsigned int norm;
} Input;

typedef struct
{
  GMappedFile **file[MAX_NINPUTS];
  char **template[MAX_NINPUTS];
  char **experiment;
  char **label;
  gsl_rng *rng;
  GeneticVariable *genetic_variable;
  FILE *file_result;
  FILE *file_variables;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  double *value;
  double *rangemin;
  double *rangemax;
  double *rangeminabs;
  double *rangemaxabs;
  double *error_best;
  double *weight;
  double *step;
  double *direction;
  double *value_old;
  double *error_old;
  unsigned int *precision;
  unsigned int *nsweeps;
  unsigned int *thread;
  unsigned int *thread_direction;
  unsigned int *simulation_best;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  double calculation_time;
  double p;
  double thresold;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int ninputs;
  unsigned int nsimulations;
  unsigned int nsteps;
  unsigned int nestimates;
  unsigned int algorithm;
  unsigned int nstart;
  unsigned int nend;
  unsigned int nstart_direction;
  unsigned int nend_direction;
  unsigned int niterations;
  unsigned int nbest;
  unsigned int nsaveds;
  unsigned int stop;
#if HAVE_MPI
  int mpi_rank;
#endif
} Optimize;

typedef struct
{
  unsigned int thread;
} ParallelData;

// Global variables
extern int ntasks;
extern unsigned int nthreads;
extern unsigned int nthreads_direction;
extern GMutex mutex[1];
extern void (*optimize_algorithm) ();
extern double (*optimize_estimate_direction) (unsigned int variable,
                                              unsigned int estimate);
extern double (*optimize_norm) (unsigned int simulation);
extern Input input[1];
extern Optimize optimize[1];
extern const xmlChar *result_name;
extern const xmlChar *variables_name;
extern const xmlChar *template[MAX_NINPUTS];
extern const char *format[NPRECISIONS];
extern const double precision[NPRECISIONS];

// Public functions
void input_new ();
void input_free ();
int input_open (char *filename);
void optimize_input (unsigned int simulation, char *input,
                     GMappedFile * template);
double optimize_parse (unsigned int simulation, unsigned int experiment);
double optimize_norm_euclidian (unsigned int simulation);
double optimize_norm_maximum (unsigned int simulation);
double optimize_norm_p (unsigned int simulation);
double optimize_norm_taxicab (unsigned int simulation);
void optimize_print ();
void optimize_save_variables (unsigned int simulation, double error);
void optimize_best (unsigned int simulation, double value);
void optimize_sequential ();
void *optimize_thread (ParallelData * data);
void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
                     double *error_best);
#if HAVE_MPI
void optimize_synchronise ();
#endif
void optimize_sweep ();
void optimize_MonteCarlo ();
void optimize_best_direction (unsigned int simulation, double value);
void optimize_direction_sequential ();
void *optimize_direction_thread (ParallelData * data);
double optimize_estimate_direction_random (unsigned int variable,
                                           unsigned int estimate);
double optimize_estimate_direction_coordinates (unsigned int
variable,
                                                unsigned int estimate);
void optimize_step_direction (unsigned int simulation);
void optimize_direction ();
double optimize_genetic_objective (Entity * entity);
void optimize_genetic ();
void optimize_save_old ();
void optimize_merge_old ();
void optimize_refine ();
void optimize_step ();
void optimize_iterate ();
void optimize_free ();
void optimize_open ();

#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

Copyright 2012-2016, all rights reserved.

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.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

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.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

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)
```

```
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

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.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

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.

```
{
  unsigned int i;
  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

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#endif
#include <libintl.h>
#if HAVE_GTK
#include <gtk/gtk.h>
#endif
#include "utils.h"

#if HAVE_GTK
GtkWindow *main_window;
#endif

void
show_message (char *title, char *msg, int type)
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *) gtk_message_dialog_new
    (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}

void
show_error (char *msg)
{
  show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
}

int
xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
                                unsigned int default_value, int *error_code)
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}

double
xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}

double
xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
                                 double default_value, int *error_code)
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}

void
xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%d", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%u", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%.14lg", value);
  xmlSetProp (node, prop, buffer);
}

int
cores_number ()
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}

#if HAVE_GTK

unsigned int
gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}

#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.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

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.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

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.

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

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.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *) gtk_message_dialog_new
    (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // 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.

```
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

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.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

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.

```
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

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.

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

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.

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

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

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2016, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef UTILS__H
#define UTILS__H 1

#if HAVE_GTK
#define ERROR_TYPE GTK_MESSAGE_ERROR
#define INFO_TYPE GTK_MESSAGE_INFO
extern GtkWindow *main_window;
#else
#define ERROR_TYPE 0
#define INFO_TYPE 0
#endif

// Public functions
void show_message (char *title, char *msg, int type);
void show_error (char *msg);
int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
                                int *error_code);
unsigned int xml_node_get_uint_with_default (xmlNode * node,
                                             const xmlChar * prop,
                                             unsigned int default_value,
                                             int *error_code);
double xml_node_get_float (xmlNode * node, const xmlChar * prop,
                           int *error_code);
double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
,
                                        double default_value, int *error_code);
void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
                        unsigned int value);
void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
int cores_number ();
#if HAVE_GTK
```

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

#endif
```

Index