MPCOTool

2.2.1

Generated by Doxygen 1.8.9.1

Mon Feb 1 2016 18:57:45

Contents

Chapter 1

MPCOTool

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

VERSIONS

- 2.2.1: Stable and recommended version.
- 2.3.2: 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:

- 2.2.1/configure.ac: configure generator.
- 2.2.1/Makefile.in: Makefile generator.
- 2.2.1/config.h.in: config header generator.
- 2.2.1/mpcotool.c: main source code.
- 2.2.1/mpcotool.h: main header code.
- 2.2.1/interface.h: interface header code.
- 2.2.1/build: script to build all.
- 2.2.1/logo.png: logo figure.
- 2.2.1/Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- README.md: this file.
- license.md: license 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/2.2.1
   $ ln -s ../../genetic/2.0.0 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/2.2.1):

```
$ cd ../tests/test2
$ ln -s ../../../genetic/2.0.0 genetic
$ cd ../test3
$ ln -s ../../../genetic/2.0.0 genetic
$ cd ../test4
$ ln -s ../../../genetic/2.0.0 genetic
```

2. Build all tests doing in the same terminal:

```
$ cd ../../2.2.1
$ make tests
```

USER INSTRUCTIONS

Optional arguments are typed in square brackets.

- Command line in sequential mode (where X is the number of threads to execute and S is a seed for the pseudo-random numbers generator):

```
$ ./mpcotoolbin [-nthreads X] [-seed S] input_file.xml [result_file] [variables_file]
```

- Command line in parallelized mode (where X is the number of threads to open for every node and S is a seed for the pseudo-random numbers generator):

```
$ mpirun [MPI options] ./mpcotoolbin [-nthreads X] [-seed S] input_file.xml [result_file] [variables↩
_file]
```

- 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"?> <optimize 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" direction="direction_search_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"> </optimize> “‘

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 direction search method by using:
  - *direction*: method to estimate the optimal direction. 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 optimal direction.
      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 direction search method,
  - relaxation: relaxation parameter,

  and for each variable:

  - step: initial step size for the direction search 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"?> <optimize 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"> </optimize> “‘

- 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 the experiment data.

`#include <experiment.h>`

Data Fields

- char * name
  File name.
- char * template [MAX_NINPUTS]
  Array of template names of input files.
- double weight
  Objective function weight.
- unsigned int ninputs
  Number of input files to the simulator.

4.1.1 Detailed Description

Struct to define the experiment data.

Definition at line 45 of file experiment.h.

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

- experiment.h

4.2 Input Struct Reference

Struct to define the optimization input file.

`#include <input.h>`

Collaboration diagram for Input:

Data Fields

- Experiment * experiment

Array or experiments.

- Variable * variable

  Array of variables.
- 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 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 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 71 of file input.h.

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

- input.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 ** 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 * nbits

  Array of bits number of the genetic 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 45 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 80 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 48 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 122 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.
- GtkSpinner * spinner

  Animation GtkSpinner.
- GtkGrid * grid

  Grid GtkGrid.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 66 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 the variable data.

`#include <variable.h>`

Data Fields

- char * name

  Variable name.
- double rangemin

  Minimum variable value.
- double rangemax

  Maximum variable value.
- double rangeminabs

  Absolute minimum variable value.
- double rangemaxabs

  Absolute maximum variable value.
- double step

  Direction search method step size.
- unsigned int precision

  Variable precision.
- unsigned int nsweeps

  Sweeps of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define the variable data.

Definition at line 56 of file variable.h.

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

- variable.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 78 of file interface.h.

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

- interface.h

Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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

5.2 config.h

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

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

5.3 experiment.c File Reference

Source file to define the experiment data.

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

Include dependency graph for experiment.c:

Macros

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

  Macro to debug.

Functions

- void experiment_new (Experiment *experiment)

  Function to create a new Experiment struct.
- void experiment_free (Experiment *experiment)

  Function to free the memory of an Experiment struct.
- void experiment_error (Experiment *experiment, char *message)

  Function to print a message error opening an Experiment struct.
- int experiment_open (Experiment *experiment, xmlNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.

Variables

- const xmlChar * template [MAX_NINPUTS]

  Array of xmlChar strings with template labels.

5.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file experiment.c.

5.3.2 Function Documentation

5.3.2.1 void experiment_error (Experiment * *experiment*, char * *message*)

Function to print a message error opening an Experiment struct.

Parameters

experiment	Experiment struct.
message	Error message.

Definition at line 109 of file experiment.c.

```
00110 {
00111   char buffer[64];
00112   if (!experiment->name)
00113     snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114   else
00115     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116               message);
00117   error_message = g_strdup (buffer);
00118 }
```

5.3.2.2 void experiment_free (Experiment * *experiment*)

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 85 of file experiment.c.

```
00086 {
00087   unsigned int i;
00088 #if DEBUG
00089   fprintf (stderr, "experiment_free: start\n");
00090 #endif
00091   for (i = 0; i < experiment->ninputs; ++i)
00092     xmlFree (experiment->template[i]);
00093   xmlFree (experiment->name);
00094   experiment->ninputs = 0;
00095 #if DEBUG
00096   fprintf (stderr, "experiment_free: end\n");
00097 #endif
00098 }
```

5.3.2.3 void experiment_new (Experiment * *experiment*)

Function to create a new Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 63 of file experiment.c.

```
00064 {
00065   unsigned int i;
00066 #if DEBUG
00067   fprintf (stderr, "experiment_new: start\n");
00068 #endif
00069   experiment->name = NULL;
00070   experiment->ninputs = 0;
00071   for (i = 0; i < MAX_NINPUTS; ++i)
00072     experiment->template[i] = NULL;
00073 #if DEBUG
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
```

5.3.2.4 int experiment_open (Experiment * *experiment*, xmlNode * *node*, unsigned int *ninputs*)

Function to open the Experiment struct on a XML node.

Parameters

experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 133 of file experiment.c.

```
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG
00140   fprintf (stderr, "experiment_open: start\n");
00141 #endif
00142
```

```
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, XML_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, gettext ("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG
00154   fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = xml_node_get_float_with_default (node,
      XML_WEIGHT, 1., &error_code);
00158   if (error_code)
00159     {
00160       experiment_error (experiment, gettext ("bad weight"));
00161       goto exit_on_error;
00162     }
00163 #if DEBUG
00164   fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00165 #endif
00166   experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00167   if (experiment->template[0])
00168     {
00169 #if DEBUG
00170       fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
00171                experiment->name, buffer2[0]);
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, gettext ("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG
00183       fprintf (stderr, "experiment_open: template%u\n", i + 1);
00184 #endif
00185       if (xmlHasProp (node, template[i]))
00186         {
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, gettext ("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00193 #if DEBUG
00194           fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
00195                    experiment->nexperiments, experiment->name,
00196                    experiment->template[i]);
00197 #endif
00198           ++experiment->ninputs;
00199         }
00200       else if (ninputs && ninputs > i)
00201         {
00202           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00203           experiment_error (experiment, buffer);
00204           goto exit_on_error;
00205         }
00206       else
00207         break;
00208     }
00209
00210 #if DEBUG
00211   fprintf (stderr, "experiment_open: end\n");
00212 #endif
00213   return 1;
00214
00215 exit_on_error:
00216   experiment_free (experiment);
00217 #if DEBUG
00218   fprintf (stderr, "experiment_open: end\n");
00219 #endif
00220   return 0;
00221 }
```

Here is the call graph for this function:

5.3.3 Variable Documentation

5.3.3.1 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 49 of file experiment.c.

5.4 experiment.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include "utils.h"
00045 #include "experiment.h"
00046
00047 #define DEBUG 0
00048
00049 const xmlChar *template[MAX_NINPUTS] = {
00050   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00051   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00052 };
00053
00055
00062 void
00063 experiment_new (Experiment * experiment)
00064 {
00065   unsigned int i;
00066 #if DEBUG
00067   fprintf (stderr, "experiment_new: start\n");
00068 #endif
00069   experiment->name = NULL;
00070   experiment->ninputs = 0;
00071   for (i = 0; i < MAX_NINPUTS; ++i)
```

```
00072     experiment->template[i] = NULL;
00073 #if DEBUG
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
00077
00084 void
00085 experiment_free (Experiment * experiment)
00086 {
00087   unsigned int i;
00088 #if DEBUG
00089   fprintf (stderr, "experiment_free: start\n");
00090 #endif
00091   for (i = 0; i < experiment->ninputs; ++i)
00092     xmlFree (experiment->template[i]);
00093   xmlFree (experiment->name);
00094   experiment->ninputs = 0;
00095 #if DEBUG
00096   fprintf (stderr, "experiment_free: end\n");
00097 #endif
00098 }
00099
00108 void
00109 experiment_error (Experiment * experiment, char *message)
00110 {
00111   char buffer[64];
00112   if (!experiment->name)
00113     snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114   else
00115     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116               message);
00117   error_message = g_strdup (buffer);
00118 }
00119
00132 int
00133 experiment_open (Experiment * experiment, xmlNode * node, unsigned int ninputs)
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG
00140   fprintf (stderr, "experiment_open: start\n");
00141 #endif
00142
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, XML_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, gettext ("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG
00154   fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = xml_node_get_float_with_default (node,
      XML_WEIGHT, 1., &error_code);
00158   if (error_code)
00159     {
00160       experiment_error (experiment, gettext ("bad weight"));
00161       goto exit_on_error;
00162     }
00163 #if DEBUG
00164   fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00165 #endif
00166   experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00167   if (experiment->template[0])
00168     {
00169 #if DEBUG
00170       fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
00171                experiment->name, buffer2[0]);
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, gettext ("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG
00183       fprintf (stderr, "experiment_open: template%u\n", i + 1);
```

```
00184 #endif
00185       if (xmlHasProp (node, template[i]))
00186         {
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, gettext ("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00193 #if DEBUG
00194           fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
00195                    experiment->nexperiments, experiment->name,
00196                    experiment->template[i]);
00197 #endif
00198           ++experiment->ninputs;
00199         }
00200       else if (ninputs && ninputs > i)
00201         {
00202           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00203           experiment_error (experiment, buffer);
00204           goto exit_on_error;
00205         }
00206       else
00207         break;
00208     }
00209
00210 #if DEBUG
00211   fprintf (stderr, "experiment_open: end\n");
00212 #endif
00213   return 1;
00214
00215 exit_on_error:
00216   experiment_free (experiment);
00217 #if DEBUG
00218   fprintf (stderr, "experiment_open: end\n");
00219 #endif
00220   return 0;
00221 }
```

5.5 experiment.h File Reference

Header file to define the experiment data.

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

Data Structures

- struct Experiment

  Struct to define the experiment data.

Functions

- void experiment_new (Experiment *experiment)

  Function to create a new Experiment struct.
- void experiment_free (Experiment *experiment)

  Function to free the memory of an Experiment struct.
- void experiment_error (Experiment *experiment, char *message)

  Function to print a message error opening an Experiment struct.
- int experiment_open (Experiment *experiment, xmlNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.

Variables

- const xmlChar * template [MAX_NINPUTS]

  Array of xmlChar strings with template labels.

5.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Definition in file experiment.h.

5.5.2 Function Documentation

5.5.2.1 void experiment_error (Experiment * *experiment*, char * *message*)

Function to print a message error opening an Experiment struct.

Parameters

experiment	Experiment struct.
message	Error message.

Definition at line 109 of file experiment.c.

```
00110 {
00111   char buffer[64];
00112   if (!experiment->name)
00113     snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114   else
00115     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116               message);
00117   error_message = g_strdup (buffer);
00118 }
```

5.5.2.2 void experiment_free (Experiment * *experiment*)

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 85 of file experiment.c.

```
00086 {
00087   unsigned int i;
00088 #if DEBUG
00089   fprintf (stderr, "experiment_free: start\n");
00090 #endif
00091   for (i = 0; i < experiment->ninputs; ++i)
00092     xmlFree (experiment->template[i]);
00093   xmlFree (experiment->name);
00094   experiment->ninputs = 0;
00095 #if DEBUG
00096   fprintf (stderr, "experiment_free: end\n");
00097 #endif
00098 }
```

5.5.2.3 void experiment_new (Experiment * *experiment*)

Function to create a new Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 63 of file experiment.c.

```
00064 {
00065   unsigned int i;
00066 #if DEBUG
00067   fprintf (stderr, "experiment_new: start\n");
00068 #endif
00069   experiment->name = NULL;
00070   experiment->ninputs = 0;
00071   for (i = 0; i < MAX_NINPUTS; ++i)
00072     experiment->template[i] = NULL;
00073 #if DEBUG
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
```

5.5.2.4 int experiment_open (Experiment * *experiment*, xmlNode * *node*, unsigned int *ninputs*)

Function to open the Experiment struct on a XML node.

Parameters

experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 133 of file experiment.c.

```
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG
00140   fprintf (stderr, "experiment_open: start\n");
00141 #endif
00142
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, XML_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, gettext ("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG
00154   fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = xml_node_get_float_with_default (node,
      XML_WEIGHT, 1., &error_code);
00158   if (error_code)
00159     {
00160       experiment_error (experiment, gettext ("bad weight"));
00161       goto exit_on_error;
00162     }
00163 #if DEBUG
00164   fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00165 #endif
00166   experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00167   if (experiment->template[0])
00168     {
00169 #if DEBUG
00170       fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
00171                experiment->name, buffer2[0]);
```

```
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, gettext ("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG
00183       fprintf (stderr, "experiment_open: template%u\n", i + 1);
00184 #endif
00185       if (xmlHasProp (node, template[i]))
00186         {
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, gettext ("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00193 #if DEBUG
00194           fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
00195                    experiment->nexperiments, experiment->name,
00196                    experiment->template[i]);
00197 #endif
00198           ++experiment->ninputs;
00199         }
00200       else if (ninputs && ninputs > i)
00201         {
00202           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00203           experiment_error (experiment, buffer);
00204           goto exit_on_error;
00205         }
00206       else
00207         break;
00208     }
00209
00210 #if DEBUG
00211   fprintf (stderr, "experiment_open: end\n");
00212 #endif
00213   return 1;
00214
00215 exit_on_error:
00216   experiment_free (experiment);
00217 #if DEBUG
00218   fprintf (stderr, "experiment_open: end\n");
00219 #endif
00220   return 0;
00221 }
```

Here is the call graph for this function:

5.6 experiment.h

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

```
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef EXPERIMENT__H
00039 #define EXPERIMENT__H 1
00040
00045 typedef struct
00046 {
00047   char *name;
00048   char *template[MAX_NINPUTS];
00049   double weight;
00050   unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const xmlChar *template[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_new (Experiment * experiment);
00057 void experiment_free (Experiment * experiment);
00058 void experiment_error (Experiment * experiment, char *message);
00059 int experiment_open (Experiment * experiment, xmlNode * node,
00060                      unsigned int ninputs);
00061
00062 #endif
```

5.7 input.c File Reference

Source file to define the input functions.

```
#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
```

Include dependency graph for input.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG_INPUT 1

  Macro to debug.

Functions

- void input_new ()

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

  Function to free the memory of the input file data.
- void input_error (char *message)

  Function to print an error message opening an Input struct.
- int input_open (char *filename)

  Function to open the input file.

Variables

- Input **input** [1]
- const xmlChar * result_name = (xmlChar *) "result"

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

  Name of the variables file.

5.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file input.c.

5.7.2 Function Documentation

5.7.2.1 void input_error (char * *message*)

Function to print an error message opening an Input struct.

Parameters

message	Error message.

Definition at line 114 of file input.c.

```
00115 {
00116   char buffer[64];
00117   snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00118   error_message = g_strdup (buffer);
00119 }
```

5.7.2.2 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 129 of file input.c.

```
00130 {
00131   char buffer2[64];
00132   xmlDoc *doc;
00133   xmlNode *node, *child;
00134   xmlChar *buffer;
```

```
00135   int error_code;
00136   unsigned int i;
00137
00138 #if DEBUG_INPUT
00139   fprintf (stderr, "input_open: start\n");
00140 #endif
00141
00142   // Resetting input data
00143   buffer = NULL;
00144   input_new ();
00145
00146   // Parsing the input file
00147 #if DEBUG_INPUT
00148   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00149 #endif
00150   doc = xmlParseFile (filename);
00151   if (!doc)
00152     {
00153       input_error (gettext ("Unable to parse the input file"));
00154       goto exit_on_error;
00155     }
00156
00157   // Getting the root node
00158 #if DEBUG_INPUT
00159   fprintf (stderr, "input_open: getting the root node\n");
00160 #endif
00161   node = xmlDocGetRootElement (doc);
00162   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00163     {
00164       input_error (gettext ("Bad root XML node"));
00165       goto exit_on_error;
00166     }
00167
00168   // Getting result and variables file names
00169   if (!input->result)
00170     {
00171       input->result = (char *) xmlGetProp (node, XML_RESULT);
00172       if (!input->result)
00173         input->result = (char *) xmlStrdup (result_name);
00174     }
00175   if (!input->variables)
00176     {
00177       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00178       if (!input->variables)
00179         input->variables = (char *) xmlStrdup (variables_name);
00180     }
00181
00182   // Opening simulator program name
00183   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00184   if (!input->simulator)
00185     {
00186       input_error (gettext ("Bad simulator program"));
00187       goto exit_on_error;
00188     }
00189
00190   // Opening evaluator program name
00191   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00192
00193   // Obtaining pseudo-random numbers generator seed
00194   input->seed
00195     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00196                                       &error_code);
00197   if (error_code)
00198     {
00199       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00200       goto exit_on_error;
00201     }
00202
00203   // Opening algorithm
00204   buffer = xmlGetProp (node, XML_ALGORITHM);
00205   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00206     {
00207       input->algorithm = ALGORITHM_MONTE_CARLO;
00208
00209       // Obtaining simulations number
00210       input->nsimulations
00211         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00212       if (error_code)
00213         {
00214           input_error (gettext ("Bad simulations number"));
00215           goto exit_on_error;
00216         }
00217     }
00218   else if (!xmlStrcmp (buffer, XML_SWEEP))
00219     input->algorithm = ALGORITHM_SWEEP;
00220   else if (!xmlStrcmp (buffer, XML_GENETIC))
```

```
00221     {
00222       input->algorithm = ALGORITHM_GENETIC;
00223
00224       // Obtaining population
00225       if (xmlHasProp (node, XML_NPOPULATION))
00226         {
00227           input->nsimulations
00228             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00229           if (error_code || input->nsimulations < 3)
00230             {
00231               input_error (gettext ("Invalid population number"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           input_error (gettext ("No population number"));
00238           goto exit_on_error;
00239         }
00240
00241       // Obtaining generations
00242       if (xmlHasProp (node, XML_NGENERATIONS))
00243         {
00244           input->niterations
00245             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00246           if (error_code || !input->niterations)
00247             {
00248               input_error (gettext ("Invalid generations number"));
00249               goto exit_on_error;
00250             }
00251         }
00252       else
00253         {
00254           input_error (gettext ("No generations number"));
00255           goto exit_on_error;
00256         }
00257
00258       // Obtaining mutation probability
00259       if (xmlHasProp (node, XML_MUTATION))
00260         {
00261           input->mutation_ratio
00262             = xml_node_get_float (node, XML_MUTATION, &error_code);
00263           if (error_code || input->mutation_ratio < 0.
00264               || input->mutation_ratio >= 1.)
00265             {
00266               input_error (gettext ("Invalid mutation probability"));
00267               goto exit_on_error;
00268             }
00269         }
00270       else
00271         {
00272           input_error (gettext ("No mutation probability"));
00273           goto exit_on_error;
00274         }
00275
00276       // Obtaining reproduction probability
00277       if (xmlHasProp (node, XML_REPRODUCTION))
00278         {
00279           input->reproduction_ratio
00280             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00281           if (error_code || input->reproduction_ratio < 0.
00282               || input->reproduction_ratio >= 1.0)
00283             {
00284               input_error (gettext ("Invalid reproduction probability"));
00285               goto exit_on_error;
00286             }
00287         }
00288       else
00289         {
00290           input_error (gettext ("No reproduction probability"));
00291           goto exit_on_error;
00292         }
00293
00294       // Obtaining adaptation probability
00295       if (xmlHasProp (node, XML_ADAPTATION))
00296         {
00297           input->adaptation_ratio
00298             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00299           if (error_code || input->adaptation_ratio < 0.
00300               || input->adaptation_ratio >= 1.)
00301             {
00302               input_error (gettext ("Invalid adaptation probability"));
00303               goto exit_on_error;
00304             }
00305         }
00306       else
00307         {
```

```
00308           input_error (gettext ("No adaptation probability"));
00309           goto exit_on_error;
00310         }
00311
00312       // Checking survivals
00313       i = input->mutation_ratio * input->nsimulations;
00314       i += input->reproduction_ratio * input->nsimulations;
00315       i += input->adaptation_ratio * input->nsimulations;
00316       if (i > input->nsimulations - 2)
00317         {
00318           input_error (gettext
00319             ("No enough survival entities to reproduce the population"));
00320           goto exit_on_error;
00321         }
00322     }
00323   else
00324     {
00325       input_error (gettext ("Unknown algorithm"));
00326       goto exit_on_error;
00327     }
00328   xmlFree (buffer);
00329   buffer = NULL;
00330
00331   if (input->algorithm == ALGORITHM_MONTE_CARLO
00332       || input->algorithm == ALGORITHM_SWEEP)
00333     {
00334
00335       // Obtaining iterations number
00336       input->niterations
00337         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00338       if (error_code == 1)
00339         input->niterations = 1;
00340       else if (error_code)
00341         {
00342           input_error (gettext ("Bad iterations number"));
00343           goto exit_on_error;
00344         }
00345
00346       // Obtaining best number
00347       input->nbest
00348         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00349       if (error_code || !input->nbest)
00350         {
00351           input_error (gettext ("Invalid best number"));
00352           goto exit_on_error;
00353         }
00354
00355       // Obtaining tolerance
00356       input->tolerance
00357         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00358                                          &error_code);
00359       if (error_code || input->tolerance < 0.)
00360         {
00361           input_error (gettext ("Invalid tolerance"));
00362           goto exit_on_error;
00363         }
00364
00365       // Getting direction search method parameters
00366       if (xmlHasProp (node, XML_NSTEPS))
00367         {
00368           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00369           if (error_code || !input->nsteps)
00370             {
00371               input_error (gettext ("Invalid steps number"));
00372               goto exit_on_error;
00373             }
00374           buffer = xmlGetProp (node, XML_DIRECTION);
00375           if (!xmlStrcmp (buffer, XML_COORDINATES))
00376             input->direction = DIRECTION_METHOD_COORDINATES;
00377           else if (!xmlStrcmp (buffer, XML_RANDOM))
00378             {
00379               input->direction = DIRECTION_METHOD_RANDOM;
00380               input->nestimates
00381                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00382               if (error_code || !input->nestimates)
00383                 {
00384                   input_error (gettext ("Invalid estimates number"));
00385                   goto exit_on_error;
00386                 }
00387             }
00388           else
00389             {
00390               input_error
00391                 (gettext ("Unknown method to estimate the direction search"));
```

```
00392               goto exit_on_error;
00393             }
00394           xmlFree (buffer);
00395           buffer = NULL;
00396           input->relaxation
00397             = xml_node_get_float_with_default (node,
    XML_RELAXATION,
00398                                                DEFAULT_RELAXATION, &error_code);
00399           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00400             {
00401               input_error (gettext ("Invalid relaxation parameter"));
00402               goto exit_on_error;
00403             }
00404         }
00405       else
00406         input->nsteps = 0;
00407     }
00408   // Obtaining the thresold
00409   input->thresold = xml_node_get_float_with_default (node,
    XML_THRESOLD, 0.,
00410                                                      &error_code);
00411   if (error_code)
00412     {
00413       input_error (gettext ("Invalid thresold"));
00414       goto exit_on_error;
00415     }
00416
00417   // Reading the experimental data
00418   for (child = node->children; child; child = child->next)
00419     {
00420       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00421         break;
00422 #if DEBUG_INPUT
00423       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00424 #endif
00425       input->experiment = (Experiment *)
00426         g_realloc (input->experiment,
00427                    (1 + input->nexperiments) * sizeof (Experiment));
00428       if (!input->nexperiments)
00429         {
00430           if (!experiment_open (input->experiment, child, 0))
00431             goto exit_on_error;
00432         }
00433       else
00434         {
00435           if (!experiment_open (input->experiment + input->
    nexperiments, child,
00436                                 input->experiment->ninputs))
00437             goto exit_on_error;
00438         }
00439       ++input->nexperiments;
00440 #if DEBUG_INPUT
00441       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00442 #endif
00443     }
00444   if (!input->nexperiments)
00445     {
00446       input_error (gettext ("No optimization experiments"));
00447       goto exit_on_error;
00448     }
00449   buffer = NULL;
00450
00451   // Reading the variables data
00452   for (; child; child = child->next)
00453     {
00454 #if DEBUG_INPUT
00455       fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);
00456 #endif
00457       if (xmlStrcmp (child->name, XML_VARIABLE))
00458         {
00459           snprintf (buffer2, 64, "%s %u: %s",
00460                     gettext ("Variable"),
00461                     input->nvariables + 1, gettext ("bad XML node"));
00462           input_error (buffer2);
00463           goto exit_on_error;
00464         }
00465       input->variable = (Variable *)
00466         g_realloc (input->variable,
00467                    (1 + input->nvariables) * sizeof (Variable));
00468       if (!variable_open (input->variable + input->
    nvariables, child,
00469                           input->algorithm, input->nsteps))
00470         goto exit_on_error;
00471       ++input->nvariables;
00472     }
00473   if (!input->nvariables)
```

```
00474     {
00475       input_error (gettext ("No optimization variables"));
00476       goto exit_on_error;
00477     }
00478   buffer = NULL;
00479
00480   // Obtaining the error norm
00481   if (xmlHasProp (node, XML_NORM))
00482     {
00483       buffer = xmlGetProp (node, XML_NORM);
00484       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00485         input->norm = ERROR_NORM_EUCLIDIAN;
00486       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00487         input->norm = ERROR_NORM_MAXIMUM;
00488       else if (!xmlStrcmp (buffer, XML_P))
00489         {
00490           input->norm = ERROR_NORM_P;
00491           input->p = xml_node_get_float (node, XML_P, &error_code);
00492           if (!error_code)
00493             {
00494               input_error (gettext ("Bad P parameter"));
00495               goto exit_on_error;
00496             }
00497         }
00498       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00499         input->norm = ERROR_NORM_TAXICAB;
00500       else
00501         {
00502           input_error (gettext ("Unknown error norm"));
00503           goto exit_on_error;
00504         }
00505       xmlFree (buffer);
00506     }
00507   else
00508     input->norm = ERROR_NORM_EUCLIDIAN;
00509
00510   // Getting the working directory
00511   input->directory = g_path_get_dirname (filename);
00512   input->name = g_path_get_basename (filename);
00513
00514   // Closing the XML document
00515   xmlFreeDoc (doc);
00516
00517 #if DEBUG_INPUT
00518   fprintf (stderr, "input_open: end\n");
00519 #endif
00520   return 1;
00521
00522 exit_on_error:
00523   xmlFree (buffer);
00524   xmlFreeDoc (doc);
00525   show_error (error_message);
00526   g_free (error_message);
00527   input_free ();
00528 #if DEBUG_INPUT
00529   fprintf (stderr, "input_open: end\n");
00530 #endif
00531   return 0;
00532 }
```

Here is the call graph for this function:

5.8 input.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
```

```
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <glib/gstdio.h>
00045 #include "utils.h"
00046 #include "experiment.h"
00047 #include "variable.h"
00048 #include "input.h"
00049
00050 #define DEBUG_INPUT 1
00051
00052 Input input[1];
00053
00054 const xmlChar *result_name = (xmlChar *) "result";
00056 const xmlChar *variables_name = (xmlChar *) "variables";
00058
00063 void
00064 input_new ()
00065 {
00066 #if DEBUG_INPUT
00067   fprintf (stderr, "input_new: start\n");
00068 #endif
00069   input->nvariables = input->nexperiments = input->nsteps = 0;
00070   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00071   input->experiment = NULL;
00072   input->variable = NULL;
00073 #if DEBUG_INPUT
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085   unsigned int i;
00086 #if DEBUG_INPUT
00087   fprintf (stderr, "input_free: start\n");
00088 #endif
00089   g_free (input->name);
00090   g_free (input->directory);
00091   for (i = 0; i < input->nexperiments; ++i)
00092     experiment_free (input->experiment + i);
00093   g_free (input->experiment);
00094   for (i = 0; i < input->nvariables; ++i)
00095     variable_free (input->variable + i);
00096   g_free (input->variable);
00097   xmlFree (input->evaluator);
00098   xmlFree (input->simulator);
00099   xmlFree (input->result);
00100   xmlFree (input->variables);
00101   input->nexperiments = input->nvariables = input->nsteps = 0;
00102 #if DEBUG_INPUT
00103   fprintf (stderr, "input_free: end\n");
00104 #endif
00105 }
00106
00113 void
00114 input_error (char *message)
00115 {
00116   char buffer[64];
00117   snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00118   error_message = g_strdup (buffer);
00119 }
00120
00128 int
00129 input_open (char *filename)
00130 {
00131   char buffer2[64];
00132   xmlDoc *doc;
00133   xmlNode *node, *child;
```

```
00134   xmlChar *buffer;
00135   int error_code;
00136   unsigned int i;
00137 
00138 #if DEBUG_INPUT
00139   fprintf (stderr, "input_open: start\n");
00140 #endif
00141 
00142   // Resetting input data
00143   buffer = NULL;
00144   input_new ();
00145 
00146   // Parsing the input file
00147 #if DEBUG_INPUT
00148   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00149 #endif
00150   doc = xmlParseFile (filename);
00151   if (!doc)
00152     {
00153       input_error (gettext ("Unable to parse the input file"));
00154       goto exit_on_error;
00155     }
00156 
00157   // Getting the root node
00158 #if DEBUG_INPUT
00159   fprintf (stderr, "input_open: getting the root node\n");
00160 #endif
00161   node = xmlDocGetRootElement (doc);
00162   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00163     {
00164       input_error (gettext ("Bad root XML node"));
00165       goto exit_on_error;
00166     }
00167 
00168   // Getting result and variables file names
00169   if (!input->result)
00170     {
00171       input->result = (char *) xmlGetProp (node, XML_RESULT);
00172       if (!input->result)
00173         input->result = (char *) xmlStrdup (result_name);
00174     }
00175   if (!input->variables)
00176     {
00177       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00178       if (!input->variables)
00179         input->variables = (char *) xmlStrdup (variables_name);
00180     }
00181 
00182   // Opening simulator program name
00183   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00184   if (!input->simulator)
00185     {
00186       input_error (gettext ("Bad simulator program"));
00187       goto exit_on_error;
00188     }
00189 
00190   // Opening evaluator program name
00191   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00192 
00193   // Obtaining pseudo-random numbers generator seed
00194   input->seed
00195     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00196                                       &error_code);
00197   if (error_code)
00198     {
00199       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00200       goto exit_on_error;
00201     }
00202 
00203   // Opening algorithm
00204   buffer = xmlGetProp (node, XML_ALGORITHM);
00205   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00206     {
00207       input->algorithm = ALGORITHM_MONTE_CARLO;
00208 
00209       // Obtaining simulations number
00210       input->nsimulations
00211         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00212       if (error_code)
00213         {
00214           input_error (gettext ("Bad simulations number"));
00215           goto exit_on_error;
00216         }
00217     }
00218   else if (!xmlStrcmp (buffer, XML_SWEEP))
00219     input->algorithm = ALGORITHM_SWEEP;
```

```
00220   else if (!xmlStrcmp (buffer, XML_GENETIC))
00221     {
00222       input->algorithm = ALGORITHM_GENETIC;
00223
00224       // Obtaining population
00225       if (xmlHasProp (node, XML_NPOPULATION))
00226         {
00227           input->nsimulations
00228             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00229           if (error_code || input->nsimulations < 3)
00230             {
00231               input_error (gettext ("Invalid population number"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           input_error (gettext ("No population number"));
00238           goto exit_on_error;
00239         }
00240
00241       // Obtaining generations
00242       if (xmlHasProp (node, XML_NGENERATIONS))
00243         {
00244           input->niterations
00245             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00246           if (error_code || !input->niterations)
00247             {
00248               input_error (gettext ("Invalid generations number"));
00249               goto exit_on_error;
00250             }
00251         }
00252       else
00253         {
00254           input_error (gettext ("No generations number"));
00255           goto exit_on_error;
00256         }
00257
00258       // Obtaining mutation probability
00259       if (xmlHasProp (node, XML_MUTATION))
00260         {
00261           input->mutation_ratio
00262             = xml_node_get_float (node, XML_MUTATION, &error_code);
00263           if (error_code || input->mutation_ratio < 0.
00264               || input->mutation_ratio >= 1.)
00265             {
00266               input_error (gettext ("Invalid mutation probability"));
00267               goto exit_on_error;
00268             }
00269         }
00270       else
00271         {
00272           input_error (gettext ("No mutation probability"));
00273           goto exit_on_error;
00274         }
00275
00276       // Obtaining reproduction probability
00277       if (xmlHasProp (node, XML_REPRODUCTION))
00278         {
00279           input->reproduction_ratio
00280             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00281           if (error_code || input->reproduction_ratio < 0.
00282               || input->reproduction_ratio >= 1.0)
00283             {
00284               input_error (gettext ("Invalid reproduction probability"));
00285               goto exit_on_error;
00286             }
00287         }
00288       else
00289         {
00290           input_error (gettext ("No reproduction probability"));
00291           goto exit_on_error;
00292         }
00293
00294       // Obtaining adaptation probability
00295       if (xmlHasProp (node, XML_ADAPTATION))
00296         {
00297           input->adaptation_ratio
00298             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00299           if (error_code || input->adaptation_ratio < 0.
00300               || input->adaptation_ratio >= 1.)
00301             {
00302               input_error (gettext ("Invalid adaptation probability"));
00303               goto exit_on_error;
00304             }
00305         }
00306       else
```

```
00307         {
00308           input_error (gettext ("No adaptation probability"));
00309           goto exit_on_error;
00310         }
00311
00312       // Checking survivals
00313       i = input->mutation_ratio * input->nsimulations;
00314       i += input->reproduction_ratio * input->nsimulations;
00315       i += input->adaptation_ratio * input->nsimulations;
00316       if (i > input->nsimulations - 2)
00317         {
00318           input_error (gettext
00319             ("No enough survival entities to reproduce the population"));
00320           goto exit_on_error;
00321         }
00322     }
00323   else
00324     {
00325       input_error (gettext ("Unknown algorithm"));
00326       goto exit_on_error;
00327     }
00328   xmlFree (buffer);
00329   buffer = NULL;
00330
00331   if (input->algorithm == ALGORITHM_MONTE_CARLO
00332       || input->algorithm == ALGORITHM_SWEEP)
00333     {
00334
00335       // Obtaining iterations number
00336       input->niterations
00337         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00338       if (error_code == 1)
00339         input->niterations = 1;
00340       else if (error_code)
00341         {
00342           input_error (gettext ("Bad iterations number"));
00343           goto exit_on_error;
00344         }
00345
00346       // Obtaining best number
00347       input->nbest
00348         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00349       if (error_code || !input->nbest)
00350         {
00351           input_error (gettext ("Invalid best number"));
00352           goto exit_on_error;
00353         }
00354
00355       // Obtaining tolerance
00356       input->tolerance
00357         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00358                                          &error_code);
00359       if (error_code || input->tolerance < 0.)
00360         {
00361           input_error (gettext ("Invalid tolerance"));
00362           goto exit_on_error;
00363         }
00364
00365       // Getting direction search method parameters
00366       if (xmlHasProp (node, XML_NSTEPS))
00367         {
00368           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00369           if (error_code || !input->nsteps)
00370             {
00371               input_error (gettext ("Invalid steps number"));
00372               goto exit_on_error;
00373             }
00374           buffer = xmlGetProp (node, XML_DIRECTION);
00375           if (!xmlStrcmp (buffer, XML_COORDINATES))
00376             input->direction = DIRECTION_METHOD_COORDINATES;
00377           else if (!xmlStrcmp (buffer, XML_RANDOM))
00378             {
00379               input->direction = DIRECTION_METHOD_RANDOM;
00380               input->nestimates
00381                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00382               if (error_code || !input->nestimates)
00383                 {
00384                   input_error (gettext ("Invalid estimates number"));
00385                   goto exit_on_error;
00386                 }
00387             }
00388           else
00389             {
00390               input_error
```

```
00391                 (gettext ("Unknown method to estimate the direction search"));
00392               goto exit_on_error;
00393             }
00394           xmlFree (buffer);
00395           buffer = NULL;
00396           input->relaxation
00397             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00398                                                DEFAULT_RELAXATION, &error_code);
00399           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00400             {
00401               input_error (gettext ("Invalid relaxation parameter"));
00402               goto exit_on_error;
00403             }
00404         }
00405       else
00406         input->nsteps = 0;
00407     }
00408   // Obtaining the thresold
00409   input->thresold = xml_node_get_float_with_default (node,
      XML_THRESOLD, 0.,
00410                                                      &error_code);
00411   if (error_code)
00412     {
00413       input_error (gettext ("Invalid thresold"));
00414       goto exit_on_error;
00415     }
00416
00417   // Reading the experimental data
00418   for (child = node->children; child; child = child->next)
00419     {
00420       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00421         break;
00422 #if DEBUG_INPUT
00423       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00424 #endif
00425       input->experiment = (Experiment *)
00426         g_realloc (input->experiment,
00427                    (1 + input->nexperiments) * sizeof (Experiment));
00428       if (!input->nexperiments)
00429         {
00430           if (!experiment_open (input->experiment, child, 0))
00431             goto exit_on_error;
00432         }
00433       else
00434         {
00435           if (!experiment_open (input->experiment + input->
      nexperiments, child,
00436                                 input->experiment->ninputs))
00437             goto exit_on_error;
00438         }
00439       ++input->nexperiments;
00440 #if DEBUG_INPUT
00441       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00442 #endif
00443     }
00444   if (!input->nexperiments)
00445     {
00446       input_error (gettext ("No optimization experiments"));
00447       goto exit_on_error;
00448     }
00449   buffer = NULL;
00450
00451   // Reading the variables data
00452   for (; child; child = child->next)
00453     {
00454 #if DEBUG_INPUT
00455       fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);
00456 #endif
00457       if (xmlStrcmp (child->name, XML_VARIABLE))
00458         {
00459           snprintf (buffer2, 64, "%s %u: %s",
00460                     gettext ("Variable"),
00461                     input->nvariables + 1, gettext ("bad XML node"));
00462           input_error (buffer2);
00463           goto exit_on_error;
00464         }
00465       input->variable = (Variable *)
00466         g_realloc (input->variable,
00467                    (1 + input->nvariables) * sizeof (Variable));
00468       if (!variable_open (input->variable + input->
      nvariables, child,
00469                           input->algorithm, input->nsteps))
00470         goto exit_on_error;
00471       ++input->nvariables;
00472     }
```

```
00473   if (!input->nvariables)
00474     {
00475       input_error (gettext ("No optimization variables"));
00476       goto exit_on_error;
00477     }
00478   buffer = NULL;
00479
00480   // Obtaining the error norm
00481   if (xmlHasProp (node, XML_NORM))
00482     {
00483       buffer = xmlGetProp (node, XML_NORM);
00484       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00485         input->norm = ERROR_NORM_EUCLIDIAN;
00486       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00487         input->norm = ERROR_NORM_MAXIMUM;
00488       else if (!xmlStrcmp (buffer, XML_P))
00489         {
00490           input->norm = ERROR_NORM_P;
00491           input->p = xml_node_get_float (node, XML_P, &error_code);
00492           if (!error_code)
00493             {
00494               input_error (gettext ("Bad P parameter"));
00495               goto exit_on_error;
00496             }
00497         }
00498       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00499         input->norm = ERROR_NORM_TAXICAB;
00500       else
00501         {
00502           input_error (gettext ("Unknown error norm"));
00503           goto exit_on_error;
00504         }
00505       xmlFree (buffer);
00506     }
00507   else
00508     input->norm = ERROR_NORM_EUCLIDIAN;
00509
00510   // Getting the working directory
00511   input->directory = g_path_get_dirname (filename);
00512   input->name = g_path_get_basename (filename);
00513
00514   // Closing the XML document
00515   xmlFreeDoc (doc);
00516
00517 #if DEBUG_INPUT
00518   fprintf (stderr, "input_open: end\n");
00519 #endif
00520   return 1;
00521
00522 exit_on_error:
00523   xmlFree (buffer);
00524   xmlFreeDoc (doc);
00525   show_error (error_message);
00526   g_free (error_message);
00527   input_free ();
00528 #if DEBUG_INPUT
00529   fprintf (stderr, "input_open: end\n");
00530 #endif
00531   return 0;
00532 }
```

5.9 input.h File Reference

Header file to define the input functions.

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

Data Structures

- struct Input

  Struct to define the optimization input file.

Enumerations

- 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.
- void input_error (char *message)

  Function to print an error message opening an Input struct.
- int input_open (char *filename)

  Function to open the input file.

Variables

- Input **input** [1]
- const xmlChar * result_name

  Name of the result file.
- const xmlChar * variables_name

  Name of the variables file.

5.9.1 Detailed Description

Header file to define the input functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file input.h.

5.9.2 Enumeration Type Documentation

5.9.2.1 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 45 of file input.h.

```
00046 {
00047   DIRECTION_METHOD_COORDINATES = 0,
00048   DIRECTION_METHOD_RANDOM = 1,
00049 };
```

5.9.2.2 enum ErrorNorm

Enum to define the error norm.

Enumerator

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

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

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

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

Definition at line 55 of file input.h.

```
00056 {
00057   ERROR_NORM_EUCLIDIAN = 0,
00059   ERROR_NORM_MAXIMUM = 1,
00061   ERROR_NORM_P = 2,
00063   ERROR_NORM_TAXICAB = 3
00065 };
```

5.9.3 Function Documentation

5.9.3.1 void input_error (char * *message*)

Function to print an error message opening an Input struct.

Parameters

message	Error message.

Definition at line 114 of file input.c.

```
00115 {
00116   char buffer[64];
00117   snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00118   error_message = g_strdup (buffer);
00119 }
```

5.9.3.2 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 129 of file input.c.

```
00130 {
00131   char buffer2[64];
00132   xmlDoc *doc;
00133   xmlNode *node, *child;
00134   xmlChar *buffer;
00135   int error_code;
00136   unsigned int i;
00137
00138 #if DEBUG_INPUT
00139   fprintf (stderr, "input_open: start\n");
00140 #endif
00141
00142   // Resetting input data
00143   buffer = NULL;
00144   input_new ();
00145
00146   // Parsing the input file
00147 #if DEBUG_INPUT
00148   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00149 #endif
00150   doc = xmlParseFile (filename);
00151   if (!doc)
00152     {
00153       input_error (gettext ("Unable to parse the input file"));
00154       goto exit_on_error;
00155     }
00156
00157   // Getting the root node
00158 #if DEBUG_INPUT
00159   fprintf (stderr, "input_open: getting the root node\n");
00160 #endif
00161   node = xmlDocGetRootElement (doc);
00162   if (xmlStrcmp (node->name, XML_OPTIMIZE))
00163     {
00164       input_error (gettext ("Bad root XML node"));
00165       goto exit_on_error;
00166     }
00167
00168   // Getting result and variables file names
00169   if (!input->result)
00170     {
00171       input->result = (char *) xmlGetProp (node, XML_RESULT);
00172       if (!input->result)
00173         input->result = (char *) xmlStrdup (result_name);
00174     }
00175   if (!input->variables)
00176     {
00177       input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00178       if (!input->variables)
00179         input->variables = (char *) xmlStrdup (variables_name);
00180     }
00181
00182   // Opening simulator program name
00183   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00184   if (!input->simulator)
00185     {
00186       input_error (gettext ("Bad simulator program"));
00187       goto exit_on_error;
00188     }
00189
00190   // Opening evaluator program name
00191   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00192
00193   // Obtaining pseudo-random numbers generator seed
00194   input->seed
00195     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00196                                       &error_code);
00197   if (error_code)
00198     {
00199       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00200       goto exit_on_error;
00201     }
00202
00203   // Opening algorithm
00204   buffer = xmlGetProp (node, XML_ALGORITHM);
00205   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00206     {
00207       input->algorithm = ALGORITHM_MONTE_CARLO;
00208
00209       // Obtaining simulations number
00210       input->nsimulations
00211         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00212       if (error_code)
00213         {
00214           input_error (gettext ("Bad simulations number"));
00215           goto exit_on_error;
```

```
00216         }
00217     }
00218   else if (!xmlStrcmp (buffer, XML_SWEEP))
00219     input->algorithm = ALGORITHM_SWEEP;
00220   else if (!xmlStrcmp (buffer, XML_GENETIC))
00221     {
00222       input->algorithm = ALGORITHM_GENETIC;
00223
00224       // Obtaining population
00225       if (xmlHasProp (node, XML_NPOPULATION))
00226         {
00227           input->nsimulations
00228             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00229           if (error_code || input->nsimulations < 3)
00230             {
00231               input_error (gettext ("Invalid population number"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           input_error (gettext ("No population number"));
00238           goto exit_on_error;
00239         }
00240
00241       // Obtaining generations
00242       if (xmlHasProp (node, XML_NGENERATIONS))
00243         {
00244           input->niterations
00245             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00246           if (error_code || !input->niterations)
00247             {
00248               input_error (gettext ("Invalid generations number"));
00249               goto exit_on_error;
00250             }
00251         }
00252       else
00253         {
00254           input_error (gettext ("No generations number"));
00255           goto exit_on_error;
00256         }
00257
00258       // Obtaining mutation probability
00259       if (xmlHasProp (node, XML_MUTATION))
00260         {
00261           input->mutation_ratio
00262             = xml_node_get_float (node, XML_MUTATION, &error_code);
00263           if (error_code || input->mutation_ratio < 0.
00264               || input->mutation_ratio >= 1.)
00265             {
00266               input_error (gettext ("Invalid mutation probability"));
00267               goto exit_on_error;
00268             }
00269         }
00270       else
00271         {
00272           input_error (gettext ("No mutation probability"));
00273           goto exit_on_error;
00274         }
00275
00276       // Obtaining reproduction probability
00277       if (xmlHasProp (node, XML_REPRODUCTION))
00278         {
00279           input->reproduction_ratio
00280             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00281           if (error_code || input->reproduction_ratio < 0.
00282               || input->reproduction_ratio >= 1.0)
00283             {
00284               input_error (gettext ("Invalid reproduction probability"));
00285               goto exit_on_error;
00286             }
00287         }
00288       else
00289         {
00290           input_error (gettext ("No reproduction probability"));
00291           goto exit_on_error;
00292         }
00293
00294       // Obtaining adaptation probability
00295       if (xmlHasProp (node, XML_ADAPTATION))
00296         {
00297           input->adaptation_ratio
00298             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00299           if (error_code || input->adaptation_ratio < 0.
00300               || input->adaptation_ratio >= 1.)
00301             {
00302               input_error (gettext ("Invalid adaptation probability"));
```

```
00303               goto exit_on_error;
00304             }
00305         }
00306       else
00307         {
00308           input_error (gettext ("No adaptation probability"));
00309           goto exit_on_error;
00310         }
00311
00312       // Checking survivals
00313       i = input->mutation_ratio * input->nsimulations;
00314       i += input->reproduction_ratio * input->nsimulations;
00315       i += input->adaptation_ratio * input->nsimulations;
00316       if (i > input->nsimulations - 2)
00317         {
00318           input_error (gettext
00319             ("No enough survival entities to reproduce the population"));
00320           goto exit_on_error;
00321         }
00322     }
00323   else
00324     {
00325       input_error (gettext ("Unknown algorithm"));
00326       goto exit_on_error;
00327     }
00328   xmlFree (buffer);
00329   buffer = NULL;
00330
00331   if (input->algorithm == ALGORITHM_MONTE_CARLO
00332       || input->algorithm == ALGORITHM_SWEEP)
00333     {
00334
00335       // Obtaining iterations number
00336       input->niterations
00337         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00338       if (error_code == 1)
00339         input->niterations = 1;
00340       else if (error_code)
00341         {
00342           input_error (gettext ("Bad iterations number"));
00343           goto exit_on_error;
00344         }
00345
00346       // Obtaining best number
00347       input->nbest
00348         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00349       if (error_code || !input->nbest)
00350         {
00351           input_error (gettext ("Invalid best number"));
00352           goto exit_on_error;
00353         }
00354
00355       // Obtaining tolerance
00356       input->tolerance
00357         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00358                                          &error_code);
00359       if (error_code || input->tolerance < 0.)
00360         {
00361           input_error (gettext ("Invalid tolerance"));
00362           goto exit_on_error;
00363         }
00364
00365       // Getting direction search method parameters
00366       if (xmlHasProp (node, XML_NSTEPS))
00367         {
00368           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00369           if (error_code || !input->nsteps)
00370             {
00371               input_error (gettext ("Invalid steps number"));
00372               goto exit_on_error;
00373             }
00374           buffer = xmlGetProp (node, XML_DIRECTION);
00375           if (!xmlStrcmp (buffer, XML_COORDINATES))
00376             input->direction = DIRECTION_METHOD_COORDINATES;
00377           else if (!xmlStrcmp (buffer, XML_RANDOM))
00378             {
00379               input->direction = DIRECTION_METHOD_RANDOM;
00380               input->nestimates
00381                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00382               if (error_code || !input->nestimates)
00383                 {
00384                   input_error (gettext ("Invalid estimates number"));
00385                   goto exit_on_error;
00386                 }
```

```
00387             }
00388           else
00389             {
00390               input_error
00391                 (gettext ("Unknown method to estimate the direction search"));
00392               goto exit_on_error;
00393             }
00394           xmlFree (buffer);
00395           buffer = NULL;
00396           input->relaxation
00397             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00398                                                DEFAULT_RELAXATION, &error_code);
00399           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00400             {
00401               input_error (gettext ("Invalid relaxation parameter"));
00402               goto exit_on_error;
00403             }
00404         }
00405       else
00406         input->nsteps = 0;
00407     }
00408   // Obtaining the thresold
00409   input->thresold = xml_node_get_float_with_default (node,
      XML_THRESOLD, 0.,
00410                                                      &error_code);
00411   if (error_code)
00412     {
00413       input_error (gettext ("Invalid thresold"));
00414       goto exit_on_error;
00415     }
00416
00417   // Reading the experimental data
00418   for (child = node->children; child; child = child->next)
00419     {
00420       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00421         break;
00422 #if DEBUG_INPUT
00423       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00424 #endif
00425       input->experiment = (Experiment *)
00426         g_realloc (input->experiment,
00427                    (1 + input->nexperiments) * sizeof (Experiment));
00428       if (!input->nexperiments)
00429         {
00430           if (!experiment_open (input->experiment, child, 0))
00431             goto exit_on_error;
00432         }
00433       else
00434         {
00435           if (!experiment_open (input->experiment + input->
      nexperiments, child,
00436                                 input->experiment->ninputs))
00437             goto exit_on_error;
00438         }
00439       ++input->nexperiments;
00440 #if DEBUG_INPUT
00441       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00442 #endif
00443     }
00444   if (!input->nexperiments)
00445     {
00446       input_error (gettext ("No optimization experiments"));
00447       goto exit_on_error;
00448     }
00449   buffer = NULL;
00450
00451   // Reading the variables data
00452   for (; child; child = child->next)
00453     {
00454 #if DEBUG_INPUT
00455       fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);
00456 #endif
00457       if (xmlStrcmp (child->name, XML_VARIABLE))
00458         {
00459           snprintf (buffer2, 64, "%s %u: %s",
00460                     gettext ("Variable"),
00461                     input->nvariables + 1, gettext ("bad XML node"));
00462           input_error (buffer2);
00463           goto exit_on_error;
00464         }
00465       input->variable = (Variable *)
00466         g_realloc (input->variable,
00467                    (1 + input->nvariables) * sizeof (Variable));
00468       if (!variable_open (input->variable + input->
      nvariables, child,
```

```
00469                           input->algorithm, input->nsteps))
00470         goto exit_on_error;
00471       ++input->nvariables;
00472     }
00473   if (!input->nvariables)
00474     {
00475       input_error (gettext ("No optimization variables"));
00476       goto exit_on_error;
00477     }
00478   buffer = NULL;
00479
00480   // Obtaining the error norm
00481   if (xmlHasProp (node, XML_NORM))
00482     {
00483       buffer = xmlGetProp (node, XML_NORM);
00484       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00485         input->norm = ERROR_NORM_EUCLIDIAN;
00486       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00487         input->norm = ERROR_NORM_MAXIMUM;
00488       else if (!xmlStrcmp (buffer, XML_P))
00489         {
00490           input->norm = ERROR_NORM_P;
00491           input->p = xml_node_get_float (node, XML_P, &error_code);
00492           if (!error_code)
00493             {
00494               input_error (gettext ("Bad P parameter"));
00495               goto exit_on_error;
00496             }
00497         }
00498       else if (!xmlStrcmp (buffer, XML_TAXICAB))
00499         input->norm = ERROR_NORM_TAXICAB;
00500       else
00501         {
00502           input_error (gettext ("Unknown error norm"));
00503           goto exit_on_error;
00504         }
00505       xmlFree (buffer);
00506     }
00507   else
00508     input->norm = ERROR_NORM_EUCLIDIAN;
00509
00510   // Getting the working directory
00511   input->directory = g_path_get_dirname (filename);
00512   input->name = g_path_get_basename (filename);
00513
00514   // Closing the XML document
00515   xmlFreeDoc (doc);
00516
00517 #if DEBUG_INPUT
00518   fprintf (stderr, "input_open: end\n");
00519 #endif
00520   return 1;
00521
00522 exit_on_error:
00523   xmlFree (buffer);
00524   xmlFreeDoc (doc);
00525   show_error (error_message);
00526   g_free (error_message);
00527   input_free ();
00528 #if DEBUG_INPUT
00529   fprintf (stderr, "input_open: end\n");
00530 #endif
00531   return 0;
00532 }
```

Here is the call graph for this function:

5.10 input.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
```

```
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INPUT__H
00039 #define INPUT__H 1
00040
00045 enum DirectionMethod
00046 {
00047   DIRECTION_METHOD_COORDINATES = 0,
00048   DIRECTION_METHOD_RANDOM = 1,
00049 };
00050
00055 enum ErrorNorm
00056 {
00057   ERROR_NORM_EUCLIDIAN = 0,
00059   ERROR_NORM_MAXIMUM = 1,
00061   ERROR_NORM_P = 2,
00063   ERROR_NORM_TAXICAB = 3
00065 };
00066
00071 typedef struct
00072 {
00073   Experiment *experiment;
00074   Variable *variable;
00075   char *result;
00076   char *variables;
00077   char *simulator;
00078   char *evaluator;
00080   char *directory;
00081   char *name;
00082   double tolerance;
00083   double mutation_ratio;
00084   double reproduction_ratio;
00085   double adaptation_ratio;
00086   double relaxation;
00087   double p;
00088   double thresold;
00089   unsigned long int seed;
00091   unsigned int nvariables;
00092   unsigned int nexperiments;
00093   unsigned int nsimulations;
00094   unsigned int algorithm;
00095   unsigned int nsteps;
00097   unsigned int direction;
00098   unsigned int nestimates;
00100   unsigned int niterations;
00101   unsigned int nbest;
00102   unsigned int norm;
00103 } Input;
00104
00105 extern Input input[1];
00106 extern const xmlChar *result_name;
00107 extern const xmlChar *variables_name;
00108
00109 // Public functions
00110 void input_new ();
00111 void input_free ();
00112 void input_error (char *message);
00113 int input_open (char *filename);
00114
00115 #endif
```

5.11 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 "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
```

Include dependency graph for interface.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG_INTERFACE 1

  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.

5.11.1 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.

5.11.2 Function Documentation

5.11.2.1 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 204 of file interface.c.

```
00205 {
00206   unsigned int i, j;
00207   char *buffer;
00208   xmlDoc *doc;
00209   xmlNode *node, *child;
00210   GFile *file, *file2;
00211
00212 #if DEBUG_INTERFACE
00213   fprintf (stderr, "input_save: start\n");
00214 #endif
00215
00216   // Getting the input file directory
00217   input->name = g_path_get_basename (filename);
00218   input->directory = g_path_get_dirname (filename);
00219   file = g_file_new_for_path (input->directory);
00220
00221   // Opening the input file
00222   doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224   // Setting root XML node
00225   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226   xmlDocSetRootElement (doc, node);
00227
00228   // Adding properties to the root XML node
00229   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233   file2 = g_file_new_for_path (input->simulator);
```

```
00234   buffer = g_file_get_relative_path (file, file2);
00235   g_object_unref (file2);
00236   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237   g_free (buffer);
00238   if (input->evaluator)
00239     {
00240       file2 = g_file_new_for_path (input->evaluator);
00241       buffer = g_file_get_relative_path (file, file2);
00242       g_object_unref (file2);
00243       if (xmlStrlen ((xmlChar *) buffer))
00244         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245       g_free (buffer);
00246     }
00247   if (input->seed != DEFAULT_RANDOM_SEED)
00248     xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250   // Setting the algorithm
00251   buffer = (char *) g_malloc (64);
00252   switch (input->algorithm)
00253     {
00254     case ALGORITHM_MONTE_CARLO:
00255       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256       snprintf (buffer, 64, "%u", input->nsimulations);
00257       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258       snprintf (buffer, 64, "%u", input->niterations);
00259       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00260       snprintf (buffer, 64, "%.3lg", input->tolerance);
00261       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262       snprintf (buffer, 64, "%u", input->nbest);
00263       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264       input_save_direction (node);
00265       break;
00266     case ALGORITHM_SWEEP:
00267       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268       snprintf (buffer, 64, "%u", input->niterations);
00269       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270       snprintf (buffer, 64, "%.3lg", input->tolerance);
00271       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272       snprintf (buffer, 64, "%u", input->nbest);
00273       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274       input_save_direction (node);
00275       break;
00276     default:
00277       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278       snprintf (buffer, 64, "%u", input->nsimulations);
00279       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280       snprintf (buffer, 64, "%u", input->niterations);
00281       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288       break;
00289     }
00290   g_free (buffer);
00291   if (input->thresold != 0.)
00292     xml_node_set_float (node, XML_THRESOLD, input->
      thresold);
00293
00294   // Setting the experimental data
00295   for (i = 0; i < input->nexperiments; ++i)
00296     {
00297       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].
      name);
00299       if (input->experiment[i].weight != 1.)
00300         xml_node_set_float (child, XML_WEIGHT, input->
      experiment[i].weight);
00301       for (j = 0; j < input->experiment->ninputs; ++j)
00302         xmlSetProp (child, template[j],
00303                     (xmlChar *) input->experiment[i].template[j]);
00304     }
00305
00306   // Setting the variables data
00307   for (i = 0; i < input->nvariables; ++i)
00308     {
00309       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00310       xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
      name);
00311       xml_node_set_float (child, XML_MINIMUM, input->
      variable[i].rangemin);
00312       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00313         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
00314                             input->variable[i].rangeminabs);
00315       xml_node_set_float (child, XML_MAXIMUM, input->
```

```
      variable[i].rangemax);
00316       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00317         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
00318                             input->variable[i].rangemaxabs);
00319       if (input->variable[i].precision != DEFAULT_PRECISION)
00320         xml_node_set_uint (child, XML_PRECISION, input->
      variable[i].precision);
00321       if (input->algorithm == ALGORITHM_SWEEP)
00322         xml_node_set_uint (child, XML_NSWEEPS, input->
      variable[i].nsweeps);
00323       else if (input->algorithm == ALGORITHM_GENETIC)
00324         xml_node_set_uint (child, XML_NBITS, input->
      variable[i].nbits);
00325       if (input->nsteps)
00326         xml_node_set_float (child, XML_STEP, input->
      variable[i].step);
00327     }
00328
00329   // Saving the error norm
00330   switch (input->norm)
00331     {
00332     case ERROR_NORM_MAXIMUM:
00333       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00334       break;
00335     case ERROR_NORM_P:
00336       xmlSetProp (node, XML_NORM, XML_P);
00337       xml_node_set_float (node, XML_P, input->p);
00338       break;
00339     case ERROR_NORM_TAXICAB:
00340       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00341     }
00342
00343   // Saving the XML file
00344   xmlSaveFormatFile (filename, doc, 1);
00345
00346   // Freeing memory
00347   xmlFreeDoc (doc);
00348
00349 #if DEBUG_INTERFACE
00350   fprintf (stderr, "input_save: end\n");
00351 #endif
00352 }
```

Here is the call graph for this function:

5.11.2.2 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 172 of file interface.c.

```
00173 {
00174 #if DEBUG_INTERFACE
00175   fprintf (stderr, "input_save_direction: start\n");
00176 #endif
00177   if (input->nsteps)
00178     {
00179       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
00180       if (input->relaxation != DEFAULT_RELAXATION)
00181         xml_node_set_float (node, XML_RELAXATION, input->
      relaxation);
00182       switch (input->direction)
00183         {
00184         case DIRECTION_METHOD_COORDINATES:
00185           xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00186           break;
00187         default:
00188           xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00189           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
00190         }
00191     }
00192 #if DEBUG_INTERFACE
00193   fprintf (stderr, "input_save_direction: end\n");
00194 #endif
00195 }
```

Here is the call graph for this function:

5.11.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 461 of file interface.c.

```
00462 {
00463   unsigned int i;
00464 #if DEBUG_INTERFACE
00465   fprintf (stderr, "window_get_algorithm: start\n");
00466 #endif
00467   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00468 #if DEBUG_INTERFACE
00469   fprintf (stderr, "window_get_algorithm: %u\n", i);
00470   fprintf (stderr, "window_get_algorithm: end\n");
00471 #endif
00472   return i;
00473 }
```

Here is the call graph for this function:

5.11.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 481 of file interface.c.

```
00482 {
00483   unsigned int i;
00484 #if DEBUG_INTERFACE
00485   fprintf (stderr, "window_get_direction: start\n");
00486 #endif
00487   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00488 #if DEBUG_INTERFACE
00489   fprintf (stderr, "window_get_direction: %u\n", i);
00490   fprintf (stderr, "window_get_direction: end\n");
00491 #endif
00492   return i;
00493 }
```

Here is the call graph for this function:

5.11.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 501 of file interface.c.

```
00502 {
00503   unsigned int i;
00504 #if DEBUG_INTERFACE
00505   fprintf (stderr, "window_get_norm: start\n");
00506 #endif
00507   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00508 #if DEBUG_INTERFACE
00509   fprintf (stderr, "window_get_norm: %u\n", i);
00510   fprintf (stderr, "window_get_norm: end\n");
00511 #endif
00512   return i;
00513 }
```

Here is the call graph for this function:

5.11.2.6 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 1560 of file interface.c.

```
01561 {
01562   unsigned int i;
01563   char *buffer;
01564 #if DEBUG_INTERFACE
01565   fprintf (stderr, "window_read: start\n");
01566 #endif
01567
01568   // Reading new input file
01569   input_free ();
01570   if (!input_open (filename))
01571     {
01572 #if DEBUG_INTERFACE
01573       fprintf (stderr, "window_read: end\n");
01574 #endif
01575       return 0;
01576     }
01577
01578   // Setting GTK+ widgets data
01579   gtk_entry_set_text (window->entry_result, input->result);
01580   gtk_entry_set_text (window->entry_variables, input->
      variables);
01581   buffer = g_build_filename (input->directory, input->simulator, NULL);
01582   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01583                                  (window->button_simulator), buffer);
01584   g_free (buffer);
01585   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01586                                 (size_t) input->evaluator);
01587   if (input->evaluator)
01588     {
01589       buffer = g_build_filename (input->directory, input->evaluator, NULL);
01590       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01591                                      (window->button_evaluator), buffer);
01592       g_free (buffer);
01593     }
01594   gtk_toggle_button_set_active
01595     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01596   switch (input->algorithm)
01597     {
01598     case ALGORITHM_MONTE_CARLO:
01599       gtk_spin_button_set_value (window->spin_simulations,
01600                                  (gdouble) input->nsimulations);
01601     case ALGORITHM_SWEEP:
01602       gtk_spin_button_set_value (window->spin_iterations,
01603                                  (gdouble) input->niterations);
01604       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01605       gtk_spin_button_set_value (window->spin_tolerance, input->
```

```
      tolerance);
01606       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
01607                                     input->nsteps);
01608       if (input->nsteps)
01609         {
01610           gtk_toggle_button_set_active
01611             (GTK_TOGGLE_BUTTON (window->button_direction
01612                                 [input->direction]), TRUE);
01613           gtk_spin_button_set_value (window->spin_steps,
01614                                      (gdouble) input->nsteps);
01615           gtk_spin_button_set_value (window->spin_relaxation,
01616                                      (gdouble) input->relaxation);
01617           switch (input->direction)
01618             {
01619             case DIRECTION_METHOD_RANDOM:
01620               gtk_spin_button_set_value (window->spin_estimates,
01621                                          (gdouble) input->nestimates);
01622             }
01623         }
01624       break;
01625     default:
01626       gtk_spin_button_set_value (window->spin_population,
01627                                  (gdouble) input->nsimulations);
01628       gtk_spin_button_set_value (window->spin_generations,
01629                                  (gdouble) input->niterations);
01630       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01631       gtk_spin_button_set_value (window->spin_reproduction,
01632                                  input->reproduction_ratio);
01633       gtk_spin_button_set_value (window->spin_adaptation,
01634                                  input->adaptation_ratio);
01635     }
01636   gtk_toggle_button_set_active
01637     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01638   gtk_spin_button_set_value (window->spin_p, input->p);
01639   gtk_spin_button_set_value (window->spin_thresold, input->
      thresold);
01640   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01641   g_signal_handler_block (window->button_experiment,
01642                           window->id_experiment_name);
01643   gtk_combo_box_text_remove_all (window->combo_experiment);
01644   for (i = 0; i < input->nexperiments; ++i)
01645     gtk_combo_box_text_append_text (window->combo_experiment,
01646                                     input->experiment[i].name);
01647   g_signal_handler_unblock
01648     (window->button_experiment, window->
      id_experiment_name);
01649   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01650   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01651   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01652   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01653   gtk_combo_box_text_remove_all (window->combo_variable);
01654   for (i = 0; i < input->nvariables; ++i)
01655     gtk_combo_box_text_append_text (window->combo_variable,
01656                                     input->variable[i].name);
01657   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01658   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01659   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01660   window_set_variable ();
01661   window_update ();
01662
01663 #if DEBUG_INTERFACE
01664   fprintf (stderr, "window_read: end\n");
01665 #endif
01666   return 1;
01667 }
```

Here is the call graph for this function:

5.11.2.7 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 554 of file interface.c.

```
00555 {
00556   GtkFileChooserDialog *dlg;
00557   GtkFileFilter *filter;
00558   char *buffer;
00559
00560 #if DEBUG_INTERFACE
00561   fprintf (stderr, "window_save: start\n");
00562 #endif
00563
00564   // Opening the saving dialog
00565   dlg = (GtkFileChooserDialog *)
00566     gtk_file_chooser_dialog_new (gettext ("Save file"),
00567                                  window->window,
00568                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00569                                  gettext ("_Cancel"),
00570                                  GTK_RESPONSE_CANCEL,
00571                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00572   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00573   buffer = g_build_filename (input->directory, input->name, NULL);
00574   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00575   g_free (buffer);
00576
00577   // Adding XML filter
00578   filter = (GtkFileFilter *) gtk_file_filter_new ();
00579   gtk_file_filter_set_name (filter, "XML");
00580   gtk_file_filter_add_pattern (filter, "*.xml");
00581   gtk_file_filter_add_pattern (filter, "*.XML");
00582   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00583
00584   // If OK response then saving
00585   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00586     {
00587
00588       // Adding properties to the root XML node
00589       input->simulator = gtk_file_chooser_get_filename
00590         (GTK_FILE_CHOOSER (window->button_simulator));
00591       if (gtk_toggle_button_get_active
00592           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00593         input->evaluator = gtk_file_chooser_get_filename
00594           (GTK_FILE_CHOOSER (window->button_evaluator));
00595       else
00596         input->evaluator = NULL;
00597       input->result
00598         = (char *) xmlStrdup ((const xmlChar *)
00599                               gtk_entry_get_text (window->entry_result));
00600       input->variables
00601         = (char *) xmlStrdup ((const xmlChar *)
00602                               gtk_entry_get_text (window->entry_variables));
00603
00604       // Setting the algorithm
00605       switch (window_get_algorithm ())
00606         {
00607         case ALGORITHM_MONTE_CARLO:
00608           input->algorithm = ALGORITHM_MONTE_CARLO;
00609           input->nsimulations
00610             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00611           input->niterations
00612             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00613           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00614           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00615           window_save_direction ();
00616           break;
00617         case ALGORITHM_SWEEP:
00618           input->algorithm = ALGORITHM_SWEEP;
00619           input->niterations
00620             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00621           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00622           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00623           window_save_direction ();
00624           break;
00625         default:
00626           input->algorithm = ALGORITHM_GENETIC;
00627           input->nsimulations
00628             = gtk_spin_button_get_value_as_int (window->spin_population);
00629           input->niterations
```

```
00630             = gtk_spin_button_get_value_as_int (window->spin_generations);
00631           input->mutation_ratio
00632             = gtk_spin_button_get_value (window->spin_mutation);
00633           input->reproduction_ratio
00634             = gtk_spin_button_get_value (window->spin_reproduction);
00635           input->adaptation_ratio
00636             = gtk_spin_button_get_value (window->spin_adaptation);
00637           break;
00638         }
00639       input->norm = window_get_norm ();
00640       input->p = gtk_spin_button_get_value (window->spin_p);
00641       input->thresold = gtk_spin_button_get_value (window->
      spin_thresold);
00642
00643       // Saving the XML file
00644       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00645       input_save (buffer);
00646
00647       // Closing and freeing memory
00648       g_free (buffer);
00649       gtk_widget_destroy (GTK_WIDGET (dlg));
00650 #if DEBUG_INTERFACE
00651       fprintf (stderr, "window_save: end\n");
00652 #endif
00653       return 1;
00654     }
00655
00656   // Closing and freeing memory
00657   gtk_widget_destroy (GTK_WIDGET (dlg));
00658 #if DEBUG_INTERFACE
00659   fprintf (stderr, "window_save: end\n");
00660 #endif
00661   return 0;
00662 }
```

Here is the call graph for this function:

5.11.2.8 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 1210 of file interface.c.

```
01211 {
01212   unsigned int i, j;
01213   char *buffer;
01214   GFile *file1, *file2;
01215 #if DEBUG_INTERFACE
01216   fprintf (stderr, "window_template_experiment: start\n");
01217 #endif
01218   i = (size_t) data;
01219   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220   file1
01221     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222   file2 = g_file_new_for_path (input->directory);
01223   buffer = g_file_get_relative_path (file2, file1);
01224   input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225   g_free (buffer);
01226   g_object_unref (file2);
01227   g_object_unref (file1);
01228 #if DEBUG_INTERFACE
01229   fprintf (stderr, "window_template_experiment: end\n");
01230 #endif
01231 }
```

5.12 interface.c

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

```
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elif !defined (BSD)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #include "genetic/genetic.h"
00060 #include "utils.h"
00061 #include "experiment.h"
00062 #include "variable.h"
00063 #include "input.h"
00064 #include "optimize.h"
00065 #include "interface.h"
00066
00067 #define DEBUG_INTERFACE 1
00068
00069
00073 #ifdef G_OS_WIN32
00074 #define INPUT_FILE "test-ga-win.xml"
00075 #else
00076 #define INPUT_FILE "test-ga.xml"
00077 #endif
00078
00079 const char *logo[] = {
00080   "32 32 3 1",
00081   "     c None",
00082   ".    c #0000FF",
00083   "+    c #FF0000",
00084   "                                ",
00085   "                                ",
00086   "                                ",
00087   "      .       .       .       ",
00088   "      .       .       .       ",
00089   "      .       .       .       ",
00090   "      .       .       .       ",
00091   "      .       .     +++       ",
00092   "      .       .    +++++      ",
00093   "      .       .    +++++      ",
00094   "      .       .    +++++      ",
00095   "    +++       .     +++     +++    ",
00096   "   +++++      .       .    +++++   ",
00097   "   +++++      .       .    +++++   ",
00098   "   +++++      .       .    +++++   ",
00099   "    +++       .       .     +++    ",
00100   "      .       .       .       ",
00101   "      .       .     +++       ",
00102   "      .       .    +++++      ",
```

```
00103   "     .    +++++    .       .     ",
00104   "     .    +++++    .       .     ",
00105   "     .     +++     .       .     ",
00106   "     .      .      .       .     ",
00107   "     .      .      .       .     ",
00108   "     .      .      .       .     ",
00109   "     .      .      .       .     ",
00110   "     .      .      .       .     ",
00111   "     .      .      .       .     ",
00112   "     .      .      .       .     ",
00113   "                                 ",
00114   "                                 ",
00115   "                                 "
00116 };
00117
00118 /*
00119 const char * logo[] = {
00120 "32 32 3 1",
00121 "     c #FFFFFFFFFFFF",
00122 ".    c #00000000FFFF",
00123 "X    c #FFFF00000000",
00124 "                                 ",
00125 "                                 ",
00126 "                                 ",
00127 "     .      .      .      .     ",
00128 "     .      .      .      .     ",
00129 "     .      .      .      .     ",
00130 "     .      .      .      .     ",
00131 "     .      .     XXX     .     ",
00132 "     .      .    XXXXX    .     ",
00133 "     .      .    XXXXX    .     ",
00134 "     .      .    XXXXX    .     ",
00135 "    XXX     .     XXX    XXX    ",
00136 "   XXXXX    .      .    XXXXX   ",
00137 "   XXXXX    .      .    XXXXX   ",
00138 "   XXXXX    .      .    XXXXX   ",
00139 "    XXX     .      .     XXX    ",
00140 "     .      .      .      .     ",
00141 "     .     XXX     .      .     ",
00142 "     .    XXXXX    .      .     ",
00143 "     .    XXXXX    .      .     ",
00144 "     .    XXXXX    .      .     ",
00145 "     .     XXX     .      .     ",
00146 "     .      .      .      .     ",
00147 "     .      .      .      .     ",
00148 "     .      .      .      .     ",
00149 "     .      .      .      .     ",
00150 "     .      .      .      .     ",
00151 "     .      .      .      .     ",
00152 "     .      .      .      .     ",
00153 "                                 ",
00154 "                                 ",
00155 "                                 "};
00156 */
00157
00158 Options options[1];
00160 Running running[1];
00162 Window window[1];
00164
00171 void
00172 input_save_direction (xmlNode * node)
00173 {
00174 #if DEBUG_INTERFACE
00175   fprintf (stderr, "input_save_direction: start\n");
00176 #endif
00177   if (input->nsteps)
00178     {
00179       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
00180       if (input->relaxation != DEFAULT_RELAXATION)
00181         xml_node_set_float (node, XML_RELAXATION, input->
      relaxation);
00182       switch (input->direction)
00183         {
00184         case DIRECTION_METHOD_COORDINATES:
00185           xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00186           break;
00187         default:
00188           xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00189           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
00190         }
00191     }
00192 #if DEBUG_INTERFACE
00193   fprintf (stderr, "input_save_direction: end\n");
00194 #endif
00195 }
```

```
00196
00203 void
00204 input_save (char *filename)
00205 {
00206   unsigned int i, j;
00207   char *buffer;
00208   xmlDoc *doc;
00209   xmlNode *node, *child;
00210   GFile *file, *file2;
00211
00212 #if DEBUG_INTERFACE
00213   fprintf (stderr, "input_save: start\n");
00214 #endif
00215
00216   // Getting the input file directory
00217   input->name = g_path_get_basename (filename);
00218   input->directory = g_path_get_dirname (filename);
00219   file = g_file_new_for_path (input->directory);
00220
00221   // Opening the input file
00222   doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224   // Setting root XML node
00225   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226   xmlDocSetRootElement (doc, node);
00227
00228   // Adding properties to the root XML node
00229   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233   file2 = g_file_new_for_path (input->simulator);
00234   buffer = g_file_get_relative_path (file, file2);
00235   g_object_unref (file2);
00236   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237   g_free (buffer);
00238   if (input->evaluator)
00239     {
00240       file2 = g_file_new_for_path (input->evaluator);
00241       buffer = g_file_get_relative_path (file, file2);
00242       g_object_unref (file2);
00243       if (xmlStrlen ((xmlChar *) buffer))
00244         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245       g_free (buffer);
00246     }
00247   if (input->seed != DEFAULT_RANDOM_SEED)
00248     xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250   // Setting the algorithm
00251   buffer = (char *) g_malloc (64);
00252   switch (input->algorithm)
00253     {
00254     case ALGORITHM_MONTE_CARLO:
00255       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256       snprintf (buffer, 64, "%u", input->nsimulations);
00257       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258       snprintf (buffer, 64, "%u", input->niterations);
00259       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00260       snprintf (buffer, 64, "%.3lg", input->tolerance);
00261       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262       snprintf (buffer, 64, "%u", input->nbest);
00263       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264       input_save_direction (node);
00265       break;
00266     case ALGORITHM_SWEEP:
00267       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268       snprintf (buffer, 64, "%u", input->niterations);
00269       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270       snprintf (buffer, 64, "%.3lg", input->tolerance);
00271       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272       snprintf (buffer, 64, "%u", input->nbest);
00273       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274       input_save_direction (node);
00275       break;
00276     default:
00277       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278       snprintf (buffer, 64, "%u", input->nsimulations);
00279       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280       snprintf (buffer, 64, "%u", input->niterations);
00281       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288       break;
```

```
00289     }
00290   g_free (buffer);
00291   if (input->thresold != 0.)
00292     xml_node_set_float (node, XML_THRESOLD, input->
      thresold);
00293
00294   // Setting the experimental data
00295   for (i = 0; i < input->nexperiments; ++i)
00296     {
00297       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].
      name);
00299       if (input->experiment[i].weight != 1.)
00300         xml_node_set_float (child, XML_WEIGHT, input->
      experiment[i].weight);
00301       for (j = 0; j < input->experiment->ninputs; ++j)
00302         xmlSetProp (child, template[j],
00303                     (xmlChar *) input->experiment[i].template[j]);
00304     }
00305
00306   // Setting the variables data
00307   for (i = 0; i < input->nvariables; ++i)
00308     {
00309       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00310       xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
      name);
00311       xml_node_set_float (child, XML_MINIMUM, input->
      variable[i].rangemin);
00312       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00313         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
00314                             input->variable[i].rangeminabs);
00315       xml_node_set_float (child, XML_MAXIMUM, input->
      variable[i].rangemax);
00316       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00317         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
00318                             input->variable[i].rangemaxabs);
00319       if (input->variable[i].precision != DEFAULT_PRECISION)
00320         xml_node_set_uint (child, XML_PRECISION, input->
      variable[i].precision);
00321       if (input->algorithm == ALGORITHM_SWEEP)
00322         xml_node_set_uint (child, XML_NSWEEPS, input->
      variable[i].nsweeps);
00323       else if (input->algorithm == ALGORITHM_GENETIC)
00324         xml_node_set_uint (child, XML_NBITS, input->
      variable[i].nbits);
00325       if (input->nsteps)
00326         xml_node_set_float (child, XML_STEP, input->
      variable[i].step);
00327     }
00328
00329   // Saving the error norm
00330   switch (input->norm)
00331     {
00332     case ERROR_NORM_MAXIMUM:
00333       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00334       break;
00335     case ERROR_NORM_P:
00336       xmlSetProp (node, XML_NORM, XML_P);
00337       xml_node_set_float (node, XML_P, input->p);
00338       break;
00339     case ERROR_NORM_TAXICAB:
00340       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00341     }
00342
00343   // Saving the XML file
00344   xmlSaveFormatFile (filename, doc, 1);
00345
00346   // Freeing memory
00347   xmlFreeDoc (doc);
00348
00349 #if DEBUG_INTERFACE
00350   fprintf (stderr, "input_save: end\n");
00351 #endif
00352 }
00353
00358 void
00359 options_new ()
00360 {
00361 #if DEBUG_INTERFACE
00362   fprintf (stderr, "options_new: start\n");
00363 #endif
00364   options->label_seed = (GtkLabel *)
00365     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00366   options->spin_seed = (GtkSpinButton *)
00367     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00368   gtk_widget_set_tooltip_text
00369     (GTK_WIDGET (options->spin_seed),
```

```
00370      gettext ("Seed to init the pseudo-random numbers generator"));
00371   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00372   options->label_threads = (GtkLabel *)
00373     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00374   options->spin_threads
00375     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00376   gtk_widget_set_tooltip_text
00377     (GTK_WIDGET (options->spin_threads),
00378      gettext ("Number of threads to perform the calibration/optimization for "
00379               "the stochastic algorithm"));
00380   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00381   options->label_direction = (GtkLabel *)
00382     gtk_label_new (gettext ("Threads number for the direction search method"));
00383   options->spin_direction
00384     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00385   gtk_widget_set_tooltip_text
00386     (GTK_WIDGET (options->spin_direction),
00387      gettext ("Number of threads to perform the calibration/optimization for "
00388               "the direction search method"));
00389   gtk_spin_button_set_value (options->spin_direction,
00390                              (gdouble) nthreads_direction);
00391   options->grid = (GtkGrid *) gtk_grid_new ();
00392   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00393   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00394   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00395                    0, 1, 1, 1);
00396   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00397                    1, 1, 1, 1);
00398   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00399                    0, 2, 1, 1);
00400   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00401                    1, 2, 1, 1);
00402   gtk_widget_show_all (GTK_WIDGET (options->grid));
00403   options->dialog = (GtkDialog *)
00404     gtk_dialog_new_with_buttons (gettext ("Options"),
00405                                  window->window,
00406                                  GTK_DIALOG_MODAL,
00407                                  gettext ("_OK"), GTK_RESPONSE_OK,
00408                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
00409                                  NULL);
00410   gtk_container_add
00411     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00412      GTK_WIDGET (options->grid));
00413   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00414     {
00415       input->seed
00416         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00417       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00418       nthreads_direction
00419         = gtk_spin_button_get_value_as_int (options->spin_direction);
00420     }
00421   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00422 #if DEBUG_INTERFACE
00423   fprintf (stderr, "options_new: end\n");
00424 #endif
00425 }
00426
00431 void
00432 running_new ()
00433 {
00434 #if DEBUG_INTERFACE
00435   fprintf (stderr, "running_new: start\n");
00436 #endif
00437   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00438   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00439   running->grid = (GtkGrid *) gtk_grid_new ();
00440   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00441   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00442   running->dialog = (GtkDialog *)
00443     gtk_dialog_new_with_buttons (gettext ("Calculating"),
00444                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00445   gtk_container_add
00446     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00447      GTK_WIDGET (running->grid));
00448   gtk_spinner_start (running->spinner);
00449   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00450 #if DEBUG_INTERFACE
00451   fprintf (stderr, "running_new: end\n");
00452 #endif
00453 }
00454
00460 unsigned int
00461 window_get_algorithm ()
00462 {
00463   unsigned int i;
00464 #if DEBUG_INTERFACE
```

```
00465   fprintf (stderr, "window_get_algorithm: start\n");
00466 #endif
00467   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00468 #if DEBUG_INTERFACE
00469   fprintf (stderr, "window_get_algorithm: %u\n", i);
00470   fprintf (stderr, "window_get_algorithm: end\n");
00471 #endif
00472   return i;
00473 }
00474
00480 unsigned int
00481 window_get_direction ()
00482 {
00483   unsigned int i;
00484 #if DEBUG_INTERFACE
00485   fprintf (stderr, "window_get_direction: start\n");
00486 #endif
00487   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00488 #if DEBUG_INTERFACE
00489   fprintf (stderr, "window_get_direction: %u\n", i);
00490   fprintf (stderr, "window_get_direction: end\n");
00491 #endif
00492   return i;
00493 }
00494
00500 unsigned int
00501 window_get_norm ()
00502 {
00503   unsigned int i;
00504 #if DEBUG_INTERFACE
00505   fprintf (stderr, "window_get_norm: start\n");
00506 #endif
00507   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00508 #if DEBUG_INTERFACE
00509   fprintf (stderr, "window_get_norm: %u\n", i);
00510   fprintf (stderr, "window_get_norm: end\n");
00511 #endif
00512   return i;
00513 }
00514
00519 void
00520 window_save_direction ()
00521 {
00522 #if DEBUG_INTERFACE
00523   fprintf (stderr, "window_save_direction: start\n");
00524 #endif
00525   if (gtk_toggle_button_get_active
00526       (GTK_TOGGLE_BUTTON (window->check_direction)))
00527     {
00528       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00529       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00530       switch (window_get_direction ())
00531         {
00532         case DIRECTION_METHOD_COORDINATES:
00533           input->direction = DIRECTION_METHOD_COORDINATES;
00534           break;
00535         default:
00536           input->direction = DIRECTION_METHOD_RANDOM;
00537           input->nestimates
00538             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00539         }
00540     }
00541   else
00542     input->nsteps = 0;
00543 #if DEBUG_INTERFACE
00544   fprintf (stderr, "window_save_direction: end\n");
00545 #endif
00546 }
00547
00553 int
00554 window_save ()
00555 {
00556   GtkFileChooserDialog *dlg;
00557   GtkFileFilter *filter;
00558   char *buffer;
00559
00560 #if DEBUG_INTERFACE
00561   fprintf (stderr, "window_save: start\n");
00562 #endif
00563
00564   // Opening the saving dialog
00565   dlg = (GtkFileChooserDialog *)
00566     gtk_file_chooser_dialog_new (gettext ("Save file"),
```

```
00567                                 window->window,
00568                                 GTK_FILE_CHOOSER_ACTION_SAVE,
00569                                 gettext ("_Cancel"),
00570                                 GTK_RESPONSE_CANCEL,
00571                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00572   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00573   buffer = g_build_filename (input->directory, input->name, NULL);
00574   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00575   g_free (buffer);
00576
00577   // Adding XML filter
00578   filter = (GtkFileFilter *) gtk_file_filter_new ();
00579   gtk_file_filter_set_name (filter, "XML");
00580   gtk_file_filter_add_pattern (filter, "*.xml");
00581   gtk_file_filter_add_pattern (filter, "*.XML");
00582   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00583
00584   // If OK response then saving
00585   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00586     {
00587
00588       // Adding properties to the root XML node
00589       input->simulator = gtk_file_chooser_get_filename
00590         (GTK_FILE_CHOOSER (window->button_simulator));
00591       if (gtk_toggle_button_get_active
00592           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00593         input->evaluator = gtk_file_chooser_get_filename
00594           (GTK_FILE_CHOOSER (window->button_evaluator));
00595       else
00596         input->evaluator = NULL;
00597       input->result
00598         = (char *) xmlStrdup ((const xmlChar *)
00599                               gtk_entry_get_text (window->entry_result));
00600       input->variables
00601         = (char *) xmlStrdup ((const xmlChar *)
00602                               gtk_entry_get_text (window->entry_variables));
00603
00604       // Setting the algorithm
00605       switch (window_get_algorithm ())
00606         {
00607         case ALGORITHM_MONTE_CARLO:
00608           input->algorithm = ALGORITHM_MONTE_CARLO;
00609           input->nsimulations
00610             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00611           input->niterations
00612             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00613           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00614           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00615           window_save_direction ();
00616           break;
00617         case ALGORITHM_SWEEP:
00618           input->algorithm = ALGORITHM_SWEEP;
00619           input->niterations
00620             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00621           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00622           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00623           window_save_direction ();
00624           break;
00625         default:
00626           input->algorithm = ALGORITHM_GENETIC;
00627           input->nsimulations
00628             = gtk_spin_button_get_value_as_int (window->spin_population);
00629           input->niterations
00630             = gtk_spin_button_get_value_as_int (window->spin_generations);
00631           input->mutation_ratio
00632             = gtk_spin_button_get_value (window->spin_mutation);
00633           input->reproduction_ratio
00634             = gtk_spin_button_get_value (window->spin_reproduction);
00635           input->adaptation_ratio
00636             = gtk_spin_button_get_value (window->spin_adaptation);
00637           break;
00638         }
00639       input->norm = window_get_norm ();
00640       input->p = gtk_spin_button_get_value (window->spin_p);
00641       input->thresold = gtk_spin_button_get_value (window->spin_thresold);
00642
00643       // Saving the XML file
00644       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00645       input_save (buffer);
00646
00647       // Closing and freeing memory
00648       g_free (buffer);
00649       gtk_widget_destroy (GTK_WIDGET (dlg));
```

```
00650 #if DEBUG_INTERFACE
00651       fprintf (stderr, "window_save: end\n");
00652 #endif
00653       return 1;
00654     }
00655
00656   // Closing and freeing memory
00657   gtk_widget_destroy (GTK_WIDGET (dlg));
00658 #if DEBUG_INTERFACE
00659   fprintf (stderr, "window_save: end\n");
00660 #endif
00661   return 0;
00662 }
00663
00668 void
00669 window_run ()
00670 {
00671   unsigned int i;
00672   char *msg, *msg2, buffer[64], buffer2[64];
00673 #if DEBUG_INTERFACE
00674   fprintf (stderr, "window_run: start\n");
00675 #endif
00676   if (!window_save ())
00677     {
00678 #if DEBUG_INTERFACE
00679       fprintf (stderr, "window_run: end\n");
00680 #endif
00681       return;
00682     }
00683   running_new ();
00684   while (gtk_events_pending ())
00685     gtk_main_iteration ();
00686   optimize_open ();
00687 #if DEBUG_INTERFACE
00688   fprintf (stderr, "window_run: closing running dialog\n");
00689 #endif
00690   gtk_spinner_stop (running->spinner);
00691   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00692 #if DEBUG_INTERFACE
00693   fprintf (stderr, "window_run: displaying results\n");
00694 #endif
00695   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00696   msg2 = g_strdup (buffer);
00697   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00698     {
00699       snprintf (buffer, 64, "%s = %s\n",
00700                 input->variable[i].name, format[input->
      variable[i].precision]);
00701       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00702       msg = g_strconcat (msg2, buffer2, NULL);
00703       g_free (msg2);
00704     }
00705   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
00706             optimize->calculation_time);
00707   msg = g_strconcat (msg2, buffer, NULL);
00708   g_free (msg2);
00709   show_message (gettext ("Best result"), msg, INFO_TYPE);
00710   g_free (msg);
00711 #if DEBUG_INTERFACE
00712   fprintf (stderr, "window_run: freeing memory\n");
00713 #endif
00714   optimize_free ();
00715 #if DEBUG_INTERFACE
00716   fprintf (stderr, "window_run: end\n");
00717 #endif
00718 }
00719
00724 void
00725 window_help ()
00726 {
00727   char *buffer, *buffer2;
00728 #if DEBUG_INTERFACE
00729   fprintf (stderr, "window_help: start\n");
00730 #endif
00731   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
00732                               gettext ("user-manual.pdf"), NULL);
00733   buffer = g_filename_to_uri (buffer2, NULL, NULL);
00734   g_free (buffer2);
00735   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
00736 #if DEBUG_INTERFACE
00737   fprintf (stderr, "window_help: uri=%s\n", buffer);
00738 #endif
00739   g_free (buffer);
00740 #if DEBUG_INTERFACE
00741   fprintf (stderr, "window_help: end\n");
00742 #endif
00743 }
```

```
00744
00749 void
00750 window_about ()
00751 {
00752   static const gchar *authors[] = {
00753     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00754     "Borja Latorre Garcés <borja.latorre@csic.es>",
00755     NULL
00756   };
00757 #if DEBUG_INTERFACE
00758   fprintf (stderr, "window_about: start\n");
00759 #endif
00760   gtk_show_about_dialog
00761     (window->window,
00762      "program_name", "MPCOTool",
00763      "comments",
00764      gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
00765               "A software to perform calibrations or optimizations of "
00766               "empirical parameters"),
00767      "authors", authors,
00768      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00769      "version", "2.2.1",
00770      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
00771      "logo", window->logo,
00772      "website", "https://github.com/jburguete/mpcotool",
00773      "license-type", GTK_LICENSE_BSD, NULL);
00774 #if DEBUG_INTERFACE
00775   fprintf (stderr, "window_about: end\n");
00776 #endif
00777 }
00778
00784 void
00785 window_update_direction ()
00786 {
00787 #if DEBUG_INTERFACE
00788   fprintf (stderr, "window_update_direction: start\n");
00789 #endif
00790   gtk_widget_show (GTK_WIDGET (window->check_direction));
00791   if (gtk_toggle_button_get_active
00792       (GTK_TOGGLE_BUTTON (window->check_direction)))
00793     {
00794       gtk_widget_show (GTK_WIDGET (window->grid_direction));
00795       gtk_widget_show (GTK_WIDGET (window->label_step));
00796       gtk_widget_show (GTK_WIDGET (window->spin_step));
00797     }
00798   switch (window_get_direction ())
00799     {
00800     case DIRECTION_METHOD_COORDINATES:
00801       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
00802       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
00803       break;
00804     default:
00805       gtk_widget_show (GTK_WIDGET (window->label_estimates));
00806       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
00807     }
00808 #if DEBUG_INTERFACE
00809   fprintf (stderr, "window_update_direction: end\n");
00810 #endif
00811 }
00812
00817 void
00818 window_update ()
00819 {
00820   unsigned int i;
00821 #if DEBUG_INTERFACE
00822   fprintf (stderr, "window_update: start\n");
00823 #endif
00824   gtk_widget_set_sensitive
00825     (GTK_WIDGET (window->button_evaluator),
00826      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
00827                                    (window->check_evaluator)));
00828   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
00829   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
00830   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
00831   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
00832   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
00833   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
00834   gtk_widget_hide (GTK_WIDGET (window->label_bests));
00835   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
00836   gtk_widget_hide (GTK_WIDGET (window->label_population));
00837   gtk_widget_hide (GTK_WIDGET (window->spin_population));
00838   gtk_widget_hide (GTK_WIDGET (window->label_generations));
00839   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
00840   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
00841   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
00842   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
00843   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
```

```
00844   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
00845   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
00846   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
00847   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
00848   gtk_widget_hide (GTK_WIDGET (window->label_bits));
00849   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
00850   gtk_widget_hide (GTK_WIDGET (window->check_direction));
00851   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
00852   gtk_widget_hide (GTK_WIDGET (window->label_step));
00853   gtk_widget_hide (GTK_WIDGET (window->spin_step));
00854   gtk_widget_hide (GTK_WIDGET (window->label_p));
00855   gtk_widget_hide (GTK_WIDGET (window->spin_p));
00856   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
00857   switch (window_get_algorithm ())
00858     {
00859     case ALGORITHM_MONTE_CARLO:
00860       gtk_widget_show (GTK_WIDGET (window->label_simulations));
00861       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
00862       gtk_widget_show (GTK_WIDGET (window->label_iterations));
00863       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00864       if (i > 1)
00865         {
00866           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00867           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00868           gtk_widget_show (GTK_WIDGET (window->label_bests));
00869           gtk_widget_show (GTK_WIDGET (window->spin_bests));
00870         }
00871       window_update_direction ();
00872       break;
00873     case ALGORITHM_SWEEP:
00874       gtk_widget_show (GTK_WIDGET (window->label_iterations));
00875       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00876       if (i > 1)
00877         {
00878           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00879           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00880           gtk_widget_show (GTK_WIDGET (window->label_bests));
00881           gtk_widget_show (GTK_WIDGET (window->spin_bests));
00882         }
00883       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
00884       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
00885       gtk_widget_show (GTK_WIDGET (window->check_direction));
00886       window_update_direction ();
00887       break;
00888     default:
00889       gtk_widget_show (GTK_WIDGET (window->label_population));
00890       gtk_widget_show (GTK_WIDGET (window->spin_population));
00891       gtk_widget_show (GTK_WIDGET (window->label_generations));
00892       gtk_widget_show (GTK_WIDGET (window->spin_generations));
00893       gtk_widget_show (GTK_WIDGET (window->label_mutation));
00894       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
00895       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
00896       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
00897       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
00898       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
00899       gtk_widget_show (GTK_WIDGET (window->label_bits));
00900       gtk_widget_show (GTK_WIDGET (window->spin_bits));
00901     }
00902   gtk_widget_set_sensitive
00903     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
00904   gtk_widget_set_sensitive
00905     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
00906   for (i = 0; i < input->experiment->ninputs; ++i)
00907     {
00908       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00909       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00910       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
00911       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
00912       g_signal_handler_block
00913         (window->check_template[i], window->id_template[i]);
00914       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
00915       gtk_toggle_button_set_active
00916         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
00917       g_signal_handler_unblock
00918         (window->button_template[i], window->id_input[i]);
00919       g_signal_handler_unblock
00920         (window->check_template[i], window->id_template[i]);
00921     }
00922   if (i > 0)
00923     {
00924       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
00925       gtk_widget_set_sensitive
00926         (GTK_WIDGET (window->button_template[i - 1]),
00927          gtk_toggle_button_get_active
```

```
00928           GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
00929       }
00930     if (i < MAX_NINPUTS)
00931       {
00932         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00933         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00934         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
00935         gtk_widget_set_sensitive
00936           (GTK_WIDGET (window->button_template[i]),
00937            gtk_toggle_button_get_active
00938            GTK_TOGGLE_BUTTON (window->check_template[i]));
00939         g_signal_handler_block
00940           (window->check_template[i], window->id_template[i]);
00941         g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
00942         gtk_toggle_button_set_active
00943           (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
00944         g_signal_handler_unblock
00945           (window->button_template[i], window->id_input[i]);
00946         g_signal_handler_unblock
00947           (window->check_template[i], window->id_template[i]);
00948       }
00949     while (++i < MAX_NINPUTS)
00950       {
00951         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
00952         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
00953       }
00954     gtk_widget_set_sensitive
00955       (GTK_WIDGET (window->spin_minabs),
00956        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
00957     gtk_widget_set_sensitive
00958       (GTK_WIDGET (window->spin_maxabs),
00959        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
00960     if (window_get_norm () == ERROR_NORM_P)
00961       {
00962         gtk_widget_show (GTK_WIDGET (window->label_p));
00963         gtk_widget_show (GTK_WIDGET (window->spin_p));
00964       }
00965 #if DEBUG_INTERFACE
00966     fprintf (stderr, "window_update: end\n");
00967 #endif
00968 }
00969
00974 void
00975 window_set_algorithm ()
00976 {
00977     int i;
00978 #if DEBUG_INTERFACE
00979     fprintf (stderr, "window_set_algorithm: start\n");
00980 #endif
00981     i = window_get_algorithm ();
00982     switch (i)
00983       {
00984       case ALGORITHM_SWEEP:
00985         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00986         if (i < 0)
00987           i = 0;
00988         gtk_spin_button_set_value (window->spin_sweeps,
00989                                    (gdouble) input->variable[i].nsweeps);
00990         break;
00991       case ALGORITHM_GENETIC:
00992         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00993         if (i < 0)
00994           i = 0;
00995         gtk_spin_button_set_value (window->spin_bits,
00996                                    (gdouble) input->variable[i].nbits);
00997       }
00998     window_update ();
00999 #if DEBUG_INTERFACE
01000     fprintf (stderr, "window_set_algorithm: end\n");
01001 #endif
01002 }
01003
01008 void
01009 window_set_experiment ()
01010 {
01011     unsigned int i, j;
01012     char *buffer1, *buffer2;
01013 #if DEBUG_INTERFACE
01014     fprintf (stderr, "window_set_experiment: start\n");
01015 #endif
01016     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01017     gtk_spin_button_set_value (window->spin_weight, input->experiment[i].
      weight);
01018     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01019     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01020     g_free (buffer1);
```

```
01021   g_signal_handler_block
01022     (window->button_experiment, window->id_experiment_name);
01023   gtk_file_chooser_set_filename
01024     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01025   g_signal_handler_unblock
01026     (window->button_experiment, window->id_experiment_name);
01027   g_free (buffer2);
01028   for (j = 0; j < input->experiment->ninputs; ++j)
01029     {
01030       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01031       buffer2 = g_build_filename (input->directory,
01032                                   input->experiment[i].template[j], NULL);
01033       gtk_file_chooser_set_filename
01034         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01035       g_free (buffer2);
01036       g_signal_handler_unblock
01037         (window->button_template[j], window->id_input[j]);
01038     }
01039 #if DEBUG_INTERFACE
01040   fprintf (stderr, "window_set_experiment: end\n");
01041 #endif
01042 }
01043
01048 void
01049 window_remove_experiment ()
01050 {
01051   unsigned int i, j;
01052 #if DEBUG_INTERFACE
01053   fprintf (stderr, "window_remove_experiment: start\n");
01054 #endif
01055   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01056   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01057   gtk_combo_box_text_remove (window->combo_experiment, i);
01058   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01059   experiment_free (input->experiment + i);
01060   --input->nexperiments;
01061   for (j = i; j < input->nexperiments; ++j)
01062     memcpy (input->experiment + j, input->experiment + j + 1,
01063             sizeof (Experiment));
01064   j = input->nexperiments - 1;
01065   if (i > j)
01066     i = j;
01067   for (j = 0; j < input->experiment->ninputs; ++j)
01068     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01069   g_signal_handler_block
01070     (window->button_experiment, window->id_experiment_name);
01071   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01072   g_signal_handler_unblock
01073     (window->button_experiment, window->id_experiment_name);
01074   for (j = 0; j < input->experiment->ninputs; ++j)
01075     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01076   window_update ();
01077 #if DEBUG_INTERFACE
01078   fprintf (stderr, "window_remove_experiment: end\n");
01079 #endif
01080 }
01081
01086 void
01087 window_add_experiment ()
01088 {
01089   unsigned int i, j;
01090 #if DEBUG_INTERFACE
01091   fprintf (stderr, "window_add_experiment: start\n");
01092 #endif
01093   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01094   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01095   gtk_combo_box_text_insert_text
01096     (window->combo_experiment, i, input->experiment[i].
      name);
01097   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01098   input->experiment = (Experiment *) g_realloc
01099     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01100   for (j = input->nexperiments - 1; j > i; --j)
01101     memcpy (input->experiment + j + 1, input->experiment + j,
01102             sizeof (Experiment));
01103   input->experiment[j + 1].name
01104     = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
01105   input->experiment[j + 1].weight = input->experiment[j].
      weight;
```

```
01106   input->experiment[j + 1].ninputs = input->experiment[j].
      ninputs;
01107   for (j = 0; j < input->experiment->ninputs; ++j)
01108     input->experiment[i + 1].template[j]
01109       = (char *) xmlStrdup ((xmlChar *) input->experiment[i].template[j]);
01110   ++input->nexperiments;
01111   for (j = 0; j < input->experiment->ninputs; ++j)
01112     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01113   g_signal_handler_block
01114     (window->button_experiment, window->id_experiment_name);
01115   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01116   g_signal_handler_unblock
01117     (window->button_experiment, window->id_experiment_name);
01118   for (j = 0; j < input->experiment->ninputs; ++j)
01119     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01120   window_update ();
01121 #if DEBUG_INTERFACE
01122   fprintf (stderr, "window_add_experiment: end\n");
01123 #endif
01124 }
01125
01130 void
01131 window_name_experiment ()
01132 {
01133   unsigned int i;
01134   char *buffer;
01135   GFile *file1, *file2;
01136 #if DEBUG_INTERFACE
01137   fprintf (stderr, "window_name_experiment: start\n");
01138 #endif
01139   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01140   file1
01141     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01142   file2 = g_file_new_for_path (input->directory);
01143   buffer = g_file_get_relative_path (file2, file1);
01144   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01145   gtk_combo_box_text_remove (window->combo_experiment, i);
01146   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01147   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01148   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01149   g_free (buffer);
01150   g_object_unref (file2);
01151   g_object_unref (file1);
01152 #if DEBUG_INTERFACE
01153   fprintf (stderr, "window_name_experiment: end\n");
01154 #endif
01155 }
01156
01161 void
01162 window_weight_experiment ()
01163 {
01164   unsigned int i;
01165 #if DEBUG_INTERFACE
01166   fprintf (stderr, "window_weight_experiment: start\n");
01167 #endif
01168   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01169   input->experiment[i].weight = gtk_spin_button_get_value (window->
      spin_weight);
01170 #if DEBUG_INTERFACE
01171   fprintf (stderr, "window_weight_experiment: end\n");
01172 #endif
01173 }
01174
01180 void
01181 window_inputs_experiment ()
01182 {
01183   unsigned int j;
01184 #if DEBUG_INTERFACE
01185   fprintf (stderr, "window_inputs_experiment: start\n");
01186 #endif
01187   j = input->experiment->ninputs - 1;
01188   if (j
01189       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01190                                         (window->check_template[j])))
01191     --input->experiment->ninputs;
01192   if (input->experiment->ninputs < MAX_NINPUTS
01193       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01194                                        (window->check_template[j])))
01195     ++input->experiment->ninputs;
01196   window_update ();
01197 #if DEBUG_INTERFACE
01198   fprintf (stderr, "window_inputs_experiment: end\n");
01199 #endif
```

```
01200 }
01201
01209 void
01210 window_template_experiment (void *data)
01211 {
01212   unsigned int i, j;
01213   char *buffer;
01214   GFile *file1, *file2;
01215 #if DEBUG_INTERFACE
01216   fprintf (stderr, "window_template_experiment: start\n");
01217 #endif
01218   i = (size_t) data;
01219   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220   file1
01221     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222   file2 = g_file_new_for_path (input->directory);
01223   buffer = g_file_get_relative_path (file2, file1);
01224   input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225   g_free (buffer);
01226   g_object_unref (file2);
01227   g_object_unref (file1);
01228 #if DEBUG_INTERFACE
01229   fprintf (stderr, "window_template_experiment: end\n");
01230 #endif
01231 }
01232
01237 void
01238 window_set_variable ()
01239 {
01240   unsigned int i;
01241 #if DEBUG_INTERFACE
01242   fprintf (stderr, "window_set_variable: start\n");
01243 #endif
01244   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01245   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01246   gtk_entry_set_text (window->entry_variable, input->variable[i].
      name);
01247   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01248   gtk_spin_button_set_value (window->spin_min, input->variable[i].
      rangemin);
01249   gtk_spin_button_set_value (window->spin_max, input->variable[i].
      rangemax);
01250   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01251     {
01252       gtk_spin_button_set_value (window->spin_minabs,
01253                                  input->variable[i].rangeminabs);
01254       gtk_toggle_button_set_active
01255         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01256     }
01257   else
01258     {
01259       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01260       gtk_toggle_button_set_active
01261         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01262     }
01263   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01264     {
01265       gtk_spin_button_set_value (window->spin_maxabs,
01266                                  input->variable[i].rangemaxabs);
01267       gtk_toggle_button_set_active
01268         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01269     }
01270   else
01271     {
01272       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01273       gtk_toggle_button_set_active
01274         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01275     }
01276   gtk_spin_button_set_value (window->spin_precision,
01277                              input->variable[i].precision);
01278   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01279   if (input->nsteps)
01280     gtk_spin_button_set_value (window->spin_step, input->variable[i].
      step);
01281 #if DEBUG_INTERFACE
01282   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01283            input->variable[i].precision);
01284 #endif
01285   switch (window_get_algorithm ())
01286     {
01287     case ALGORITHM_SWEEP:
01288       gtk_spin_button_set_value (window->spin_sweeps,
01289                                  (gdouble) input->variable[i].nsweeps);
01290 #if DEBUG_INTERFACE
```

```
01291       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01292                input->variable[i].nsweeps);
01293 #endif
01294       break;
01295     case ALGORITHM_GENETIC:
01296       gtk_spin_button_set_value (window->spin_bits,
01297                                  (gdouble) input->variable[i].nbits);
01298 #if DEBUG_INTERFACE
01299       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01300                input->variable[i].nbits);
01301 #endif
01302       break;
01303     }
01304   window_update ();
01305 #if DEBUG_INTERFACE
01306   fprintf (stderr, "window_set_variable: end\n");
01307 #endif
01308 }
01309
01314 void
01315 window_remove_variable ()
01316 {
01317   unsigned int i, j;
01318 #if DEBUG_INTERFACE
01319   fprintf (stderr, "window_remove_variable: start\n");
01320 #endif
01321   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01322   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01323   gtk_combo_box_text_remove (window->combo_variable, i);
01324   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01325   xmlFree (input->variable[i].name);
01326   --input->nvariables;
01327   for (j = i; j < input->nvariables; ++j)
01328     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01329   j = input->nvariables - 1;
01330   if (i > j)
01331     i = j;
01332   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01333   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01334   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01335   window_update ();
01336 #if DEBUG_INTERFACE
01337   fprintf (stderr, "window_remove_variable: end\n");
01338 #endif
01339 }
01340
01345 void
01346 window_add_variable ()
01347 {
01348   unsigned int i, j;
01349 #if DEBUG_INTERFACE
01350   fprintf (stderr, "window_add_variable: start\n");
01351 #endif
01352   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01353   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01354   gtk_combo_box_text_insert_text (window->combo_variable, i,
01355                                   input->variable[i].name);
01356   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01357   input->variable = (Variable *) g_realloc
01358     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01359   for (j = input->nvariables - 1; j > i; --j)
01360     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01361   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01362   input->variable[j + 1].name
01363     = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01364   ++input->nvariables;
01365   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01366   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01367   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01368   window_update ();
01369 #if DEBUG_INTERFACE
01370   fprintf (stderr, "window_add_variable: end\n");
01371 #endif
01372 }
01373
```

```
01378 void
01379 window_label_variable ()
01380 {
01381   unsigned int i;
01382   const char *buffer;
01383 #if DEBUG_INTERFACE
01384   fprintf (stderr, "window_label_variable: start\n");
01385 #endif
01386   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01387   buffer = gtk_entry_get_text (window->entry_variable);
01388   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01389   gtk_combo_box_text_remove (window->combo_variable, i);
01390   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01391   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01392   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01393 #if DEBUG_INTERFACE
01394   fprintf (stderr, "window_label_variable: end\n");
01395 #endif
01396 }
01397
01402 void
01403 window_precision_variable ()
01404 {
01405   unsigned int i;
01406 #if DEBUG_INTERFACE
01407   fprintf (stderr, "window_precision_variable: start\n");
01408 #endif
01409   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01410   input->variable[i].precision
01411     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01412   gtk_spin_button_set_digits (window->spin_min, input->variable[i].
      precision);
01413   gtk_spin_button_set_digits (window->spin_max, input->variable[i].
      precision);
01414   gtk_spin_button_set_digits (window->spin_minabs,
01415                               input->variable[i].precision);
01416   gtk_spin_button_set_digits (window->spin_maxabs,
01417                               input->variable[i].precision);
01418 #if DEBUG_INTERFACE
01419   fprintf (stderr, "window_precision_variable: end\n");
01420 #endif
01421 }
01422
01427 void
01428 window_rangemin_variable ()
01429 {
01430   unsigned int i;
01431 #if DEBUG_INTERFACE
01432   fprintf (stderr, "window_rangemin_variable: start\n");
01433 #endif
01434   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01435   input->variable[i].rangemin = gtk_spin_button_get_value (window->
      spin_min);
01436 #if DEBUG_INTERFACE
01437   fprintf (stderr, "window_rangemin_variable: end\n");
01438 #endif
01439 }
01440
01445 void
01446 window_rangemax_variable ()
01447 {
01448   unsigned int i;
01449 #if DEBUG_INTERFACE
01450   fprintf (stderr, "window_rangemax_variable: start\n");
01451 #endif
01452   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01453   input->variable[i].rangemax = gtk_spin_button_get_value (window->
      spin_max);
01454 #if DEBUG_INTERFACE
01455   fprintf (stderr, "window_rangemax_variable: end\n");
01456 #endif
01457 }
01458
01463 void
01464 window_rangeminabs_variable ()
01465 {
01466   unsigned int i;
01467 #if DEBUG_INTERFACE
01468   fprintf (stderr, "window_rangeminabs_variable: start\n");
01469 #endif
01470   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01471   input->variable[i].rangeminabs
01472     = gtk_spin_button_get_value (window->spin_minabs);
01473 #if DEBUG_INTERFACE
01474   fprintf (stderr, "window_rangeminabs_variable: end\n");
```

```
01475 #endif
01476 }
01477
01482 void
01483 window_rangemaxabs_variable ()
01484 {
01485   unsigned int i;
01486 #if DEBUG_INTERFACE
01487   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01488 #endif
01489   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01490   input->variable[i].rangemaxabs
01491     = gtk_spin_button_get_value (window->spin_maxabs);
01492 #if DEBUG_INTERFACE
01493   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01494 #endif
01495 }
01496
01501 void
01502 window_step_variable ()
01503 {
01504   unsigned int i;
01505 #if DEBUG_INTERFACE
01506   fprintf (stderr, "window_step_variable: start\n");
01507 #endif
01508   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01509   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01510 #if DEBUG_INTERFACE
01511   fprintf (stderr, "window_step_variable: end\n");
01512 #endif
01513 }
01514
01519 void
01520 window_update_variable ()
01521 {
01522   int i;
01523 #if DEBUG_INTERFACE
01524   fprintf (stderr, "window_update_variable: start\n");
01525 #endif
01526   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01527   if (i < 0)
01528     i = 0;
01529   switch (window_get_algorithm ())
01530     {
01531     case ALGORITHM_SWEEP:
01532       input->variable[i].nsweeps
01533         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01534 #if DEBUG_INTERFACE
01535       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01536                input->variable[i].nsweeps);
01537 #endif
01538       break;
01539     case ALGORITHM_GENETIC:
01540       input->variable[i].nbits
01541         = gtk_spin_button_get_value_as_int (window->spin_bits);
01542 #if DEBUG_INTERFACE
01543       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01544                input->variable[i].nbits);
01545 #endif
01546     }
01547 #if DEBUG_INTERFACE
01548   fprintf (stderr, "window_update_variable: end\n");
01549 #endif
01550 }
01551
01559 int
01560 window_read (char *filename)
01561 {
01562   unsigned int i;
01563   char *buffer;
01564 #if DEBUG_INTERFACE
01565   fprintf (stderr, "window_read: start\n");
01566 #endif
01567
01568   // Reading new input file
01569   input_free ();
01570   if (!input_open (filename))
01571     {
01572 #if DEBUG_INTERFACE
01573       fprintf (stderr, "window_read: end\n");
01574 #endif
01575       return 0;
01576     }
01577
01578   // Setting GTK+ widgets data
01579   gtk_entry_set_text (window->entry_result, input->result);
```

```
01580   gtk_entry_set_text (window->entry_variables, input->variables);
01581   buffer = g_build_filename (input->directory, input->simulator, NULL);
01582   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01583                                  (window->button_simulator), buffer);
01584   g_free (buffer);
01585   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01586                                 (size_t) input->evaluator);
01587   if (input->evaluator)
01588     {
01589       buffer = g_build_filename (input->directory, input->evaluator, NULL);
01590       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01591                                      (window->button_evaluator), buffer);
01592       g_free (buffer);
01593     }
01594   gtk_toggle_button_set_active
01595     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01596   switch (input->algorithm)
01597     {
01598     case ALGORITHM_MONTE_CARLO:
01599       gtk_spin_button_set_value (window->spin_simulations,
01600                                  (gdouble) input->nsimulations);
01601     case ALGORITHM_SWEEP:
01602       gtk_spin_button_set_value (window->spin_iterations,
01603                                  (gdouble) input->niterations);
01604       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01605       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01606       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
01607                                     input->nsteps);
01608       if (input->nsteps)
01609         {
01610           gtk_toggle_button_set_active
01611             (GTK_TOGGLE_BUTTON (window->button_direction
01612                                 [input->direction]), TRUE);
01613           gtk_spin_button_set_value (window->spin_steps,
01614                                      (gdouble) input->nsteps);
01615           gtk_spin_button_set_value (window->spin_relaxation,
01616                                      (gdouble) input->relaxation);
01617           switch (input->direction)
01618             {
01619             case DIRECTION_METHOD_RANDOM:
01620               gtk_spin_button_set_value (window->spin_estimates,
01621                                          (gdouble) input->nestimates);
01622             }
01623         }
01624       break;
01625     default:
01626       gtk_spin_button_set_value (window->spin_population,
01627                                  (gdouble) input->nsimulations);
01628       gtk_spin_button_set_value (window->spin_generations,
01629                                  (gdouble) input->niterations);
01630       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01631       gtk_spin_button_set_value (window->spin_reproduction,
01632                                  input->reproduction_ratio);
01633       gtk_spin_button_set_value (window->spin_adaptation,
01634                                  input->adaptation_ratio);
01635     }
01636   gtk_toggle_button_set_active
01637     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01638   gtk_spin_button_set_value (window->spin_p, input->p);
01639   gtk_spin_button_set_value (window->spin_thresold, input->thresold);
01640   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01641   g_signal_handler_block (window->button_experiment,
01642                           window->id_experiment_name);
01643   gtk_combo_box_text_remove_all (window->combo_experiment);
01644   for (i = 0; i < input->nexperiments; ++i)
01645     gtk_combo_box_text_append_text (window->combo_experiment,
01646                                     input->experiment[i].name);
01647   g_signal_handler_unblock
01648     (window->button_experiment, window->id_experiment_name);
01649   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01650   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01651   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01652   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01653   gtk_combo_box_text_remove_all (window->combo_variable);
01654   for (i = 0; i < input->nvariables; ++i)
01655     gtk_combo_box_text_append_text (window->combo_variable,
01656                                     input->variable[i].name);
01657   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
```

```
01658   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01659   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01660   window_set_variable ();
01661   window_update ();
01662
01663 #if DEBUG_INTERFACE
01664   fprintf (stderr, "window_read: end\n");
01665 #endif
01666   return 1;
01667 }
01668
01673 void
01674 window_open ()
01675 {
01676   GtkFileChooserDialog *dlg;
01677   GtkFileFilter *filter;
01678   char *buffer, *directory, *name;
01679
01680 #if DEBUG_INTERFACE
01681   fprintf (stderr, "window_open: start\n");
01682 #endif
01683
01684   // Saving a backup of the current input file
01685   directory = g_strdup (input->directory);
01686   name = g_strdup (input->name);
01687
01688   // Opening dialog
01689   dlg = (GtkFileChooserDialog *)
01690     gtk_file_chooser_dialog_new (gettext ("Open input file"),
01691                                  window->window,
01692                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01693                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
01694                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
01695
01696   // Adding XML filter
01697   filter = (GtkFileFilter *) gtk_file_filter_new ();
01698   gtk_file_filter_set_name (filter, "XML");
01699   gtk_file_filter_add_pattern (filter, "*.xml");
01700   gtk_file_filter_add_pattern (filter, "*.XML");
01701   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
01702
01703   // If OK saving
01704   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
01705     {
01706
01707       // Traying to open the input file
01708       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
01709       if (!window_read (buffer))
01710         {
01711 #if DEBUG_INTERFACE
01712           fprintf (stderr, "window_open: error reading input file\n");
01713 #endif
01714           g_free (buffer);
01715
01716           // Reading backup file on error
01717           buffer = g_build_filename (directory, name, NULL);
01718           if (!input_open (buffer))
01719             {
01720
01721               // Closing on backup file reading error
01722 #if DEBUG_INTERFACE
01723               fprintf (stderr, "window_read: error reading backup file\n");
01724 #endif
01725               g_free (buffer);
01726               break;
01727             }
01728           g_free (buffer);
01729         }
01730       else
01731         {
01732           g_free (buffer);
01733           break;
01734         }
01735     }
01736
01737   // Freeing and closing
01738   g_free (name);
01739   g_free (directory);
01740   gtk_widget_destroy (GTK_WIDGET (dlg));
01741 #if DEBUG_INTERFACE
01742   fprintf (stderr, "window_open: end\n");
01743 #endif
01744 }
01745
01750 void
01751 window_new ()
```

```
01752 {
01753   unsigned int i;
01754   char *buffer, *buffer2, buffer3[64];
01755   char *label_algorithm[NALGORITHMS] = {
01756     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
01757   };
01758   char *tip_algorithm[NALGORITHMS] = {
01759     gettext ("Monte-Carlo brute force algorithm"),
01760     gettext ("Sweep brute force algorithm"),
01761     gettext ("Genetic algorithm")
01762   };
01763   char *label_direction[NDIRECTIONS] = {
01764     gettext ("_Coordinates descent"), gettext ("_Random")
01765   };
01766   char *tip_direction[NDIRECTIONS] = {
01767     gettext ("Coordinates direction estimate method"),
01768     gettext ("Random direction estimate method")
01769   };
01770   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
01771   char *tip_norm[NNORMS] = {
01772     gettext ("Euclidean error norm (L2)"),
01773     gettext ("Maximum error norm (L)"),
01774     gettext ("P error norm (Lp)"),
01775     gettext ("Taxicab error norm (L1)")
01776   };
01777
01778 #if DEBUG_INTERFACE
01779   fprintf (stderr, "window_new: start\n");
01780 #endif
01781
01782   // Creating the window
01783   window->window = main_window
01784     = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
01785
01786   // Finish when closing the window
01787   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
01788
01789   // Setting the window title
01790   gtk_window_set_title (window->window, "MPCOTool");
01791
01792   // Creating the open button
01793   window->button_open = (GtkToolButton *) gtk_tool_button_new
01794     (gtk_image_new_from_icon_name ("document-open",
01795                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01796      gettext ("Open"));
01797   g_signal_connect (window->button_open, "clicked", window_open, NULL);
01798
01799   // Creating the save button
01800   window->button_save = (GtkToolButton *) gtk_tool_button_new
01801     (gtk_image_new_from_icon_name ("document-save",
01802                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01803      gettext ("Save"));
01804   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
01805                     NULL);
01806
01807   // Creating the run button
01808   window->button_run = (GtkToolButton *) gtk_tool_button_new
01809     (gtk_image_new_from_icon_name ("system-run",
01810                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01811      gettext ("Run"));
01812   g_signal_connect (window->button_run, "clicked", window_run, NULL);
01813
01814   // Creating the options button
01815   window->button_options = (GtkToolButton *) gtk_tool_button_new
01816     (gtk_image_new_from_icon_name ("preferences-system",
01817                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01818      gettext ("Options"));
01819   g_signal_connect (window->button_options, "clicked", options_new, NULL);
01820
01821   // Creating the help button
01822   window->button_help = (GtkToolButton *) gtk_tool_button_new
01823     (gtk_image_new_from_icon_name ("help-browser",
01824                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01825      gettext ("Help"));
01826   g_signal_connect (window->button_help, "clicked", window_help, NULL);
01827
01828   // Creating the about button
01829   window->button_about = (GtkToolButton *) gtk_tool_button_new
01830     (gtk_image_new_from_icon_name ("help-about",
01831                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
01832      gettext ("About"));
01833   g_signal_connect (window->button_about, "clicked", window_about, NULL);
01834
01835   // Creating the exit button
01836   window->button_exit = (GtkToolButton *) gtk_tool_button_new
01837     (gtk_image_new_from_icon_name ("application-exit",
```

```
01838                                     GTK_ICON_SIZE_LARGE_TOOLBAR),
01839      gettext ("Exit"));
01840   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
01841
01842   // Creating the buttons bar
01843   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
01844   gtk_toolbar_insert
01845     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
01846   gtk_toolbar_insert
01847     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
01848   gtk_toolbar_insert
01849     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
01850   gtk_toolbar_insert
01851     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
01852   gtk_toolbar_insert
01853     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
01854   gtk_toolbar_insert
01855     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
01856   gtk_toolbar_insert
01857     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
01858   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
01859
01860   // Creating the simulator program label and entry
01861   window->label_simulator
01862     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
01863   window->button_simulator = (GtkFileChooserButton *)
01864     gtk_file_chooser_button_new (gettext ("Simulator program"),
01865                                  GTK_FILE_CHOOSER_ACTION_OPEN);
01866   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
01867                                gettext ("Simulator program executable file"));
01868   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
01869
01870   // Creating the evaluator program label and entry
01871   window->check_evaluator = (GtkCheckButton *)
01872     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
01873   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
01874   window->button_evaluator = (GtkFileChooserButton *)
01875     gtk_file_chooser_button_new (gettext ("Evaluator program"),
01876                                  GTK_FILE_CHOOSER_ACTION_OPEN);
01877   gtk_widget_set_tooltip_text
01878     (GTK_WIDGET (window->button_evaluator),
01879      gettext ("Optional evaluator program executable file"));
01880
01881   // Creating the results files labels and entries
01882   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
01883   window->entry_result = (GtkEntry *) gtk_entry_new ();
01884   gtk_widget_set_tooltip_text
01885     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
01886   window->label_variables
01887     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
01888   window->entry_variables = (GtkEntry *) gtk_entry_new ();
01889   gtk_widget_set_tooltip_text
01890     (GTK_WIDGET (window->entry_variables),
01891      gettext ("All simulated results file"));
01892
01893   // Creating the files grid and attaching widgets
01894   window->grid_files = (GtkGrid *) gtk_grid_new ();
01895   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
01896                    0, 0, 1, 1);
01897   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
01898                    1, 0, 1, 1);
01899   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
01900                    0, 1, 1, 1);
01901   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
01902                    1, 1, 1, 1);
01903   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
01904                    0, 2, 1, 1);
01905   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
01906                    1, 2, 1, 1);
01907   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
01908                    0, 3, 1, 1);
01909   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
01910                    1, 3, 1, 1);
01911
01912   // Creating the algorithm properties
01913   window->label_simulations = (GtkLabel *) gtk_label_new
01914     (gettext ("Simulations number"));
01915   window->spin_simulations
```

```
01916     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01917   gtk_widget_set_tooltip_text
01918     (GTK_WIDGET (window->spin_simulations),
01919      gettext ("Number of simulations to perform for each iteration"));
01920   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
01921   window->label_iterations = (GtkLabel *)
01922     gtk_label_new (gettext ("Iterations number"));
01923   window->spin_iterations
01924     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01925   gtk_widget_set_tooltip_text
01926     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
01927   g_signal_connect
01928     (window->spin_iterations, "value-changed", window_update, NULL);
01929   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
01930   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
01931   window->spin_tolerance
01932     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01933   gtk_widget_set_tooltip_text
01934     (GTK_WIDGET (window->spin_tolerance),
01935      gettext ("Tolerance to set the variable interval on the next iteration"));
01936   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
01937   window->spin_bests
01938     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01939   gtk_widget_set_tooltip_text
01940     (GTK_WIDGET (window->spin_bests),
01941      gettext ("Number of best simulations used to set the variable interval "
01942               "on the next iteration"));
01943   window->label_population
01944     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
01945   window->spin_population
01946     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01947   gtk_widget_set_tooltip_text
01948     (GTK_WIDGET (window->spin_population),
01949      gettext ("Number of population for the genetic algorithm"));
01950   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
01951   window->label_generations
01952     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
01953   window->spin_generations
01954     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01955   gtk_widget_set_tooltip_text
01956     (GTK_WIDGET (window->spin_generations),
01957      gettext ("Number of generations for the genetic algorithm"));
01958   window->label_mutation
01959     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
01960   window->spin_mutation
01961     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01962   gtk_widget_set_tooltip_text
01963     (GTK_WIDGET (window->spin_mutation),
01964      gettext ("Ratio of mutation for the genetic algorithm"));
01965   window->label_reproduction
01966     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
01967   window->spin_reproduction
01968     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01969   gtk_widget_set_tooltip_text
01970     (GTK_WIDGET (window->spin_reproduction),
01971      gettext ("Ratio of reproduction for the genetic algorithm"));
01972   window->label_adaptation
01973     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
01974   window->spin_adaptation
01975     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01976   gtk_widget_set_tooltip_text
01977     (GTK_WIDGET (window->spin_adaptation),
01978      gettext ("Ratio of adaptation for the genetic algorithm"));
01979   window->label_thresold = (GtkLabel *) gtk_label_new (gettext ("Thresold"));
01980   window->spin_thresold = (GtkSpinButton *) gtk_spin_button_new_with_range
01981     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
01982   gtk_widget_set_tooltip_text
01983     (GTK_WIDGET (window->spin_thresold),
01984      gettext ("Thresold in the objective function to finish the simulations"));
01985   window->scrolled_thresold
01986     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
01987   gtk_container_add (GTK_CONTAINER (window->scrolled_thresold),
01988                      GTK_WIDGET (window->spin_thresold));
01989 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_thresold), TRUE);
01990 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_thresold),
01991 //                         GTK_ALIGN_FILL);
01992
01993   // Creating the direction search method properties
01994   window->check_direction = (GtkCheckButton *)
01995     gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
01996   g_signal_connect (window->check_direction, "clicked",
    window_update, NULL);
01997   window->grid_direction = (GtkGrid *) gtk_grid_new ();
01998   window->button_direction[0] = (GtkRadioButton *)
01999     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02000   gtk_grid_attach (window->grid_direction,
02001                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
```

```
02002   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02003                     NULL);
02004   for (i = 0; ++i < NDIRECTIONS;)
02005     {
02006       window->button_direction[i] = (GtkRadioButton *)
02007         gtk_radio_button_new_with_mnemonic
02008         (gtk_radio_button_get_group (window->button_direction[0]),
02009          label_direction[i]);
02010       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02011                                    tip_direction[i]);
02012       gtk_grid_attach (window->grid_direction,
02013                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02014       g_signal_connect (window->button_direction[i], "clicked",
02015                         window_update, NULL);
02016     }
02017   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
02018   window->spin_steps = (GtkSpinButton *)
02019     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02020   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02021   window->label_estimates
02022     = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
02023   window->spin_estimates = (GtkSpinButton *)
02024     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02025   window->label_relaxation
02026     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
02027   window->spin_relaxation = (GtkSpinButton *)
02028     gtk_spin_button_new_with_range (0., 2., 0.001);
02029   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02030                    0, NDIRECTIONS, 1, 1);
02031   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02032                    1, NDIRECTIONS, 1, 1);
02033   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_estimates),
02034                    0, NDIRECTIONS + 1, 1, 1);
02035   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_estimates),
02036                    1, NDIRECTIONS + 1, 1, 1);
02037   gtk_grid_attach (window->grid_direction,
02038                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
02039                    1);
02040   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_relaxation),
02041                    1, NDIRECTIONS + 2, 1, 1);
02042
02043   // Creating the array of algorithms
02044   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02045   window->button_algorithm[0] = (GtkRadioButton *)
02046     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02047   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02048                                tip_algorithm[0]);
02049   gtk_grid_attach (window->grid_algorithm,
02050                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02051   g_signal_connect (window->button_algorithm[0], "clicked",
02052                     window_set_algorithm, NULL);
02053   for (i = 0; ++i < NALGORITHMS;)
02054     {
02055       window->button_algorithm[i] = (GtkRadioButton *)
02056         gtk_radio_button_new_with_mnemonic
02057         (gtk_radio_button_get_group (window->button_algorithm[0]),
02058          label_algorithm[i]);
02059       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02060                                    tip_algorithm[i]);
02061       gtk_grid_attach (window->grid_algorithm,
02062                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02063       g_signal_connect (window->button_algorithm[i], "clicked",
02064                         window_set_algorithm, NULL);
02065     }
02066   gtk_grid_attach (window->grid_algorithm,
02067                    GTK_WIDGET (window->label_simulations), 0,
02068                    NALGORITHMS, 1, 1);
02069   gtk_grid_attach (window->grid_algorithm,
02070                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02071   gtk_grid_attach (window->grid_algorithm,
02072                    GTK_WIDGET (window->label_iterations), 0,
02073                    NALGORITHMS + 1, 1, 1);
02074   gtk_grid_attach (window->grid_algorithm,
02075                    GTK_WIDGET (window->spin_iterations), 1,
02076                    NALGORITHMS + 1, 1, 1);
02077   gtk_grid_attach (window->grid_algorithm,
02078                    GTK_WIDGET (window->label_tolerance), 0,
02079                    NALGORITHMS + 2, 1, 1);
02080   gtk_grid_attach (window->grid_algorithm,
02081                    GTK_WIDGET (window->spin_tolerance), 1,
02082                    NALGORITHMS + 2, 1, 1);
```

```
02083   gtk_grid_attach (window->grid_algorithm,
02084                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02085   gtk_grid_attach (window->grid_algorithm,
02086                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02087   gtk_grid_attach (window->grid_algorithm,
02088                    GTK_WIDGET (window->label_population), 0,
02089                    NALGORITHMS + 4, 1, 1);
02090   gtk_grid_attach (window->grid_algorithm,
02091                    GTK_WIDGET (window->spin_population), 1,
02092                    NALGORITHMS + 4, 1, 1);
02093   gtk_grid_attach (window->grid_algorithm,
02094                    GTK_WIDGET (window->label_generations), 0,
02095                    NALGORITHMS + 5, 1, 1);
02096   gtk_grid_attach (window->grid_algorithm,
02097                    GTK_WIDGET (window->spin_generations), 1,
02098                    NALGORITHMS + 5, 1, 1);
02099   gtk_grid_attach (window->grid_algorithm,
02100                    GTK_WIDGET (window->label_mutation), 0,
02101                    NALGORITHMS + 6, 1, 1);
02102   gtk_grid_attach (window->grid_algorithm,
02103                    GTK_WIDGET (window->spin_mutation), 1,
02104                    NALGORITHMS + 6, 1, 1);
02105   gtk_grid_attach (window->grid_algorithm,
02106                    GTK_WIDGET (window->label_reproduction), 0,
02107                    NALGORITHMS + 7, 1, 1);
02108   gtk_grid_attach (window->grid_algorithm,
02109                    GTK_WIDGET (window->spin_reproduction), 1,
02110                    NALGORITHMS + 7, 1, 1);
02111   gtk_grid_attach (window->grid_algorithm,
02112                    GTK_WIDGET (window->label_adaptation), 0,
02113                    NALGORITHMS + 8, 1, 1);
02114   gtk_grid_attach (window->grid_algorithm,
02115                    GTK_WIDGET (window->spin_adaptation), 1,
02116                    NALGORITHMS + 8, 1, 1);
02117   gtk_grid_attach (window->grid_algorithm,
02118                    GTK_WIDGET (window->check_direction), 0,
02119                    NALGORITHMS + 9, 2, 1);
02120   gtk_grid_attach (window->grid_algorithm,
02121                    GTK_WIDGET (window->grid_direction), 0,
02122                    NALGORITHMS + 10, 2, 1);
02123   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_thresold),
02124                    0, NALGORITHMS + 11, 1, 1);
02125   gtk_grid_attach (window->grid_algorithm,
02126                    GTK_WIDGET (window->scrolled_thresold), 1,
02127                    NALGORITHMS + 11, 1, 1);
02128   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02129   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02130                      GTK_WIDGET (window->grid_algorithm));
02131
02132   // Creating the variable widgets
02133   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02134   gtk_widget_set_tooltip_text
02135     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02136   window->id_variable = g_signal_connect
02137     (window->combo_variable, "changed", window_set_variable, NULL);
02138   window->button_add_variable
02139     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02140                                                    GTK_ICON_SIZE_BUTTON);
02141   g_signal_connect
02142     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02143   gtk_widget_set_tooltip_text
02144     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02145   window->button_remove_variable
02146     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02147                                                    GTK_ICON_SIZE_BUTTON);
02148   g_signal_connect
02149     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02150   gtk_widget_set_tooltip_text
02151     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02152   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02153   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02154   gtk_widget_set_tooltip_text
02155     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02156   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02157   window->id_variable_label = g_signal_connect
02158     (window->entry_variable, "changed", window_label_variable, NULL);
02159   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02160   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02161     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02162   gtk_widget_set_tooltip_text
02163     (GTK_WIDGET (window->spin_min),
02164      gettext ("Minimum initial value of the variable"));
02165   window->scrolled_min
02166     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
```

```
02167   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02168                      GTK_WIDGET (window->spin_min));
02169   g_signal_connect (window->spin_min, "value-changed",
02170                     window_rangemin_variable, NULL);
02171   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02172   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02173     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02174   gtk_widget_set_tooltip_text
02175     (GTK_WIDGET (window->spin_max),
02176      gettext ("Maximum initial value of the variable"));
02177   window->scrolled_max
02178     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02179   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02180                      GTK_WIDGET (window->spin_max));
02181   g_signal_connect (window->spin_max, "value-changed",
02182                     window_rangemax_variable, NULL);
02183   window->check_minabs = (GtkCheckButton *)
02184     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
02185   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02186   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02187     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02188   gtk_widget_set_tooltip_text
02189     (GTK_WIDGET (window->spin_minabs),
02190      gettext ("Minimum allowed value of the variable"));
02191   window->scrolled_minabs
02192     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02193   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02194                      GTK_WIDGET (window->spin_minabs));
02195   g_signal_connect (window->spin_minabs, "value-changed",
02196                     window_rangeminabs_variable, NULL);
02197   window->check_maxabs = (GtkCheckButton *)
02198     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
02199   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02200   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02201     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02202   gtk_widget_set_tooltip_text
02203     (GTK_WIDGET (window->spin_maxabs),
02204      gettext ("Maximum allowed value of the variable"));
02205   window->scrolled_maxabs
02206     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02207   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02208                      GTK_WIDGET (window->spin_maxabs));
02209   g_signal_connect (window->spin_maxabs, "value-changed",
02210                     window_rangemaxabs_variable, NULL);
02211   window->label_precision
02212     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02213   window->spin_precision = (GtkSpinButton *)
02214     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02215   gtk_widget_set_tooltip_text
02216     (GTK_WIDGET (window->spin_precision),
02217      gettext ("Number of precision floating point digits\n"
02218               "0 is for integer numbers"));
02219   g_signal_connect (window->spin_precision, "value-changed",
02220                     window_precision_variable, NULL);
02221   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02222   window->spin_sweeps
02223     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02224   gtk_widget_set_tooltip_text
02225     (GTK_WIDGET (window->spin_sweeps),
02226      gettext ("Number of steps sweeping the variable"));
02227   g_signal_connect
02228     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02229   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02230   window->spin_bits
02231     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02232   gtk_widget_set_tooltip_text
02233     (GTK_WIDGET (window->spin_bits),
02234      gettext ("Number of bits to encode the variable"));
02235   g_signal_connect
02236     (window->spin_bits, "value-changed", window_update_variable, NULL);
02237   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02238   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02239     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02240   gtk_widget_set_tooltip_text
02241     (GTK_WIDGET (window->spin_step),
02242      gettext ("Initial step size for the direction search method"));
02243   window->scrolled_step
02244     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02245   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02246                      GTK_WIDGET (window->spin_step));
02247   g_signal_connect
02248     (window->spin_step, "value-changed", window_step_variable, NULL);
02249   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02250   gtk_grid_attach (window->grid_variable,
02251                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02252   gtk_grid_attach (window->grid_variable,
02253                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
```

```
02254   gtk_grid_attach (window->grid_variable,
02255                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02256   gtk_grid_attach (window->grid_variable,
02257                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02258   gtk_grid_attach (window->grid_variable,
02259                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02260   gtk_grid_attach (window->grid_variable,
02261                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02262   gtk_grid_attach (window->grid_variable,
02263                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02264   gtk_grid_attach (window->grid_variable,
02265                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02266   gtk_grid_attach (window->grid_variable,
02267                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02268   gtk_grid_attach (window->grid_variable,
02269                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02270   gtk_grid_attach (window->grid_variable,
02271                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02272   gtk_grid_attach (window->grid_variable,
02273                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02274   gtk_grid_attach (window->grid_variable,
02275                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02276   gtk_grid_attach (window->grid_variable,
02277                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02278   gtk_grid_attach (window->grid_variable,
02279                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02280   gtk_grid_attach (window->grid_variable,
02281                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02282   gtk_grid_attach (window->grid_variable,
02283                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02284   gtk_grid_attach (window->grid_variable,
02285                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02286   gtk_grid_attach (window->grid_variable,
02287                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02288   gtk_grid_attach (window->grid_variable,
02289                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02290   gtk_grid_attach (window->grid_variable,
02291                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02292   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02293   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02294                      GTK_WIDGET (window->grid_variable));
02295
02296   // Creating the experiment widgets
02297   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02298   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02299                                gettext ("Experiment selector"));
02300   window->id_experiment = g_signal_connect
02301     (window->combo_experiment, "changed", window_set_experiment, NULL)
    ;
02302   window->button_add_experiment
02303     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02304                                                    GTK_ICON_SIZE_BUTTON);
02305   g_signal_connect
02306     (window->button_add_experiment, "clicked",
    window_add_experiment, NULL);
02307   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02308                                gettext ("Add experiment"));
02309   window->button_remove_experiment
02310     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02311                                                    GTK_ICON_SIZE_BUTTON);
02312   g_signal_connect (window->button_remove_experiment, "clicked",
02313                     window_remove_experiment, NULL);
02314   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02315                                gettext ("Remove experiment"));
02316   window->label_experiment
02317     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02318   window->button_experiment = (GtkFileChooserButton *)
02319     gtk_file_chooser_button_new (gettext ("Experimental data file"),
02320                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02321   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02322                                gettext ("Experimental data file"));
02323   window->id_experiment_name
02324     = g_signal_connect (window->button_experiment, "selection-changed",
02325                         window_name_experiment, NULL);
02326   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02327   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02328   window->spin_weight
02329     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02330   gtk_widget_set_tooltip_text
02331     (GTK_WIDGET (window->spin_weight),
02332      gettext ("Weight factor to build the objective function"));
02333   g_signal_connect
02334     (window->spin_weight, "value-changed", window_weight_experiment,
    NULL);
02335   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02336   gtk_grid_attach (window->grid_experiment,
02337                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
```

```
02338   gtk_grid_attach (window->grid_experiment,
02339                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02340   gtk_grid_attach (window->grid_experiment,
02341                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02342   gtk_grid_attach (window->grid_experiment,
02343                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02344   gtk_grid_attach (window->grid_experiment,
02345                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02346   gtk_grid_attach (window->grid_experiment,
02347                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02348   gtk_grid_attach (window->grid_experiment,
02349                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02350   for (i = 0; i < MAX_NINPUTS; ++i)
02351     {
02352       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02353       window->check_template[i] = (GtkCheckButton *)
02354         gtk_check_button_new_with_label (buffer3);
02355       window->id_template[i]
02356         = g_signal_connect (window->check_template[i], "toggled",
02357                             window_inputs_experiment, NULL);
02358       gtk_grid_attach (window->grid_experiment,
02359                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02360       window->button_template[i] = (GtkFileChooserButton *)
02361         gtk_file_chooser_button_new (gettext ("Input template"),
02362                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02363       gtk_widget_set_tooltip_text
02364         (GTK_WIDGET (window->button_template[i]),
02365          gettext ("Experimental input template file"));
02366       window->id_input[i]
02367         = g_signal_connect_swapped (window->button_template[i],
02368                                     "selection-changed",
02369                                     (void (*)) window_template_experiment,
02370                                     (void *) (size_t) i);
02371       gtk_grid_attach (window->grid_experiment,
02372                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02373     }
02374   window->frame_experiment
02375     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02376   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02377                      GTK_WIDGET (window->grid_experiment));
02378
02379   // Creating the error norm widgets
02380   window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02381   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02382   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02383                      GTK_WIDGET (window->grid_norm));
02384   window->button_norm[0] = (GtkRadioButton *)
02385     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02386   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02387                                tip_norm[0]);
02388   gtk_grid_attach (window->grid_norm,
02389                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02390   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02391   for (i = 0; ++i < NNORMS;)
02392     {
02393       window->button_norm[i] = (GtkRadioButton *)
02394         gtk_radio_button_new_with_mnemonic
02395         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02396       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02397                                    tip_norm[i]);
02398       gtk_grid_attach (window->grid_norm,
02399                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02400       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02401     }
02402   window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02403   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02404   window->spin_p = (GtkSpinButton *)
02405     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02406   gtk_widget_set_tooltip_text
02407     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02408   window->scrolled_p
02409     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02410   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02411                      GTK_WIDGET (window->spin_p));
02412   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02413   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02414   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02415                    1, 2, 1, 2);
02416
02417   // Creating the grid and attaching the widgets to the grid
02418   window->grid = (GtkGrid *) gtk_grid_new ();
02419   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02420   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02421   gtk_grid_attach (window->grid,
02422                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02423   gtk_grid_attach (window->grid,
```

```
02424                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02425   gtk_grid_attach (window->grid,
02426                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02427   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02428   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
02429
02430   // Setting the window logo
02431   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02432   gtk_window_set_icon (window->window, window->logo);
02433
02434   // Showing the window
02435   gtk_widget_show_all (GTK_WIDGET (window->window));
02436
02437   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02438 #if GTK_MINOR_VERSION >= 16
02439   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02440   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02441   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02442   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02443   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02444   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02445   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_thresold), -1, 40);
02446 #endif
02447
02448   // Reading initial example
02449   input_new ();
02450   buffer2 = g_get_current_dir ();
02451   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02452   g_free (buffer2);
02453   window_read (buffer);
02454   g_free (buffer);
02455
02456 #if DEBUG_INTERFACE
02457   fprintf (stderr, "window_new: start\n");
02458 #endif
02459 }
```

5.13 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 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.

5.13.1 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.

5.13.2 Function Documentation

5.13.2.1 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 353 of file utils.c.

```
00354 {
00355   unsigned int i;
00356   for (i = 0; i < n; ++i)
00357     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00358       break;
00359   return i;
00360 }
```

5.13.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 204 of file interface.c.

```
00205 {
00206   unsigned int i, j;
00207   char *buffer;
00208   xmlDoc *doc;
00209   xmlNode *node, *child;
00210   GFile *file, *file2;
00211
00212 #if DEBUG_INTERFACE
00213   fprintf (stderr, "input_save: start\n");
00214 #endif
00215
00216   // Getting the input file directory
00217   input->name = g_path_get_basename (filename);
00218   input->directory = g_path_get_dirname (filename);
00219   file = g_file_new_for_path (input->directory);
00220
00221   // Opening the input file
00222   doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224   // Setting root XML node
00225   node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226   xmlDocSetRootElement (doc, node);
00227
00228   // Adding properties to the root XML node
00229   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233   file2 = g_file_new_for_path (input->simulator);
00234   buffer = g_file_get_relative_path (file, file2);
00235   g_object_unref (file2);
00236   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237   g_free (buffer);
00238   if (input->evaluator)
00239     {
00240       file2 = g_file_new_for_path (input->evaluator);
00241       buffer = g_file_get_relative_path (file, file2);
00242       g_object_unref (file2);
00243       if (xmlStrlen ((xmlChar *) buffer))
00244         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245       g_free (buffer);
00246     }
00247   if (input->seed != DEFAULT_RANDOM_SEED)
00248     xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250   // Setting the algorithm
00251   buffer = (char *) g_malloc (64);
00252   switch (input->algorithm)
00253     {
00254     case ALGORITHM_MONTE_CARLO:
00255       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256       snprintf (buffer, 64, "%u", input->nsimulations);
00257       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258       snprintf (buffer, 64, "%u", input->niterations);
00259       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
```

```
00260       snprintf (buffer, 64, "%.3lg", input->tolerance);
00261       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262       snprintf (buffer, 64, "%u", input->nbest);
00263       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264       input_save_direction (node);
00265       break;
00266     case ALGORITHM_SWEEP:
00267       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268       snprintf (buffer, 64, "%u", input->niterations);
00269       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270       snprintf (buffer, 64, "%.3lg", input->tolerance);
00271       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272       snprintf (buffer, 64, "%u", input->nbest);
00273       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274       input_save_direction (node);
00275       break;
00276     default:
00277       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278       snprintf (buffer, 64, "%u", input->nsimulations);
00279       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280       snprintf (buffer, 64, "%u", input->niterations);
00281       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288       break;
00289     }
00290   g_free (buffer);
00291   if (input->thresold != 0.)
00292     xml_node_set_float (node, XML_THRESOLD, input->
    thresold);
00293
00294   // Setting the experimental data
00295   for (i = 0; i < input->nexperiments; ++i)
00296     {
00297       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].
    name);
00299       if (input->experiment[i].weight != 1.)
00300         xml_node_set_float (child, XML_WEIGHT, input->
    experiment[i].weight);
00301       for (j = 0; j < input->experiment->ninputs; ++j)
00302         xmlSetProp (child, template[j],
00303                     (xmlChar *) input->experiment[i].template[j]);
00304     }
00305
00306   // Setting the variables data
00307   for (i = 0; i < input->nvariables; ++i)
00308     {
00309       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00310       xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
    name);
00311       xml_node_set_float (child, XML_MINIMUM, input->
    variable[i].rangemin);
00312       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00313         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
00314                             input->variable[i].rangeminabs);
00315       xml_node_set_float (child, XML_MAXIMUM, input->
    variable[i].rangemax);
00316       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00317         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
00318                             input->variable[i].rangemaxabs);
00319       if (input->variable[i].precision != DEFAULT_PRECISION)
00320         xml_node_set_uint (child, XML_PRECISION, input->
    variable[i].precision);
00321       if (input->algorithm == ALGORITHM_SWEEP)
00322         xml_node_set_uint (child, XML_NSWEEPS, input->
    variable[i].nsweeps);
00323       else if (input->algorithm == ALGORITHM_GENETIC)
00324         xml_node_set_uint (child, XML_NBITS, input->
    variable[i].nbits);
00325       if (input->nsteps)
00326         xml_node_set_float (child, XML_STEP, input->
    variable[i].step);
00327     }
00328
00329   // Saving the error norm
00330   switch (input->norm)
00331     {
00332     case ERROR_NORM_MAXIMUM:
00333       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00334       break;
00335     case ERROR_NORM_P:
00336       xmlSetProp (node, XML_NORM, XML_P);
```

```
00337       xml_node_set_float (node, XML_P, input->p);
00338       break;
00339     case ERROR_NORM_TAXICAB:
00340       xmlSetProp (node, XML_NORM, XML_TAXICAB);
00341     }
00342
00343   // Saving the XML file
00344   xmlSaveFormatFile (filename, doc, 1);
00345
00346   // Freeing memory
00347   xmlFreeDoc (doc);
00348
00349 #if DEBUG_INTERFACE
00350   fprintf (stderr, "input_save: end\n");
00351 #endif
00352 }
```

Here is the call graph for this function:

5.13.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 461 of file interface.c.

```
00462 {
00463   unsigned int i;
00464 #if DEBUG_INTERFACE
00465   fprintf (stderr, "window_get_algorithm: start\n");
00466 #endif
00467   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00468 #if DEBUG_INTERFACE
00469   fprintf (stderr, "window_get_algorithm: %u\n", i);
00470   fprintf (stderr, "window_get_algorithm: end\n");
00471 #endif
00472   return i;
00473 }
```

Here is the call graph for this function:

5.13.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 481 of file interface.c.

```
00482 {
00483   unsigned int i;
00484 #if DEBUG_INTERFACE
00485   fprintf (stderr, "window_get_direction: start\n");
00486 #endif
00487   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00488 #if DEBUG_INTERFACE
00489   fprintf (stderr, "window_get_direction: %u\n", i);
00490   fprintf (stderr, "window_get_direction: end\n");
00491 #endif
00492   return i;
00493 }
```

Here is the call graph for this function:

5.13.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 501 of file interface.c.

```
00502 {
00503   unsigned int i;
00504 #if DEBUG_INTERFACE
00505   fprintf (stderr, "window_get_norm: start\n");
00506 #endif
00507   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00508 #if DEBUG_INTERFACE
00509   fprintf (stderr, "window_get_norm: %u\n", i);
00510   fprintf (stderr, "window_get_norm: end\n");
00511 #endif
00512   return i;
00513 }
```

Here is the call graph for this function:

5.13.2.6 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 1560 of file interface.c.

```
01561 {
01562   unsigned int i;
01563   char *buffer;
01564 #if DEBUG_INTERFACE
01565   fprintf (stderr, "window_read: start\n");
01566 #endif
01567
01568   // Reading new input file
01569   input_free ();
01570   if (!input_open (filename))
01571     {
01572 #if DEBUG_INTERFACE
01573       fprintf (stderr, "window_read: end\n");
01574 #endif
01575       return 0;
01576     }
01577
01578   // Setting GTK+ widgets data
01579   gtk_entry_set_text (window->entry_result, input->result);
01580   gtk_entry_set_text (window->entry_variables, input->
      variables);
01581   buffer = g_build_filename (input->directory, input->simulator, NULL);
01582   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01583                                  (window->button_simulator), buffer);
01584   g_free (buffer);
01585   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01586                                 (size_t) input->evaluator);
01587   if (input->evaluator)
01588     {
01589       buffer = g_build_filename (input->directory, input->evaluator, NULL);
01590       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01591                                      (window->button_evaluator), buffer);
01592       g_free (buffer);
```

```
01593     }
01594   gtk_toggle_button_set_active
01595     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01596   switch (input->algorithm)
01597     {
01598     case ALGORITHM_MONTE_CARLO:
01599       gtk_spin_button_set_value (window->spin_simulations,
01600                                  (gdouble) input->nsimulations);
01601     case ALGORITHM_SWEEP:
01602       gtk_spin_button_set_value (window->spin_iterations,
01603                                  (gdouble) input->niterations);
01604       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01605       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01606       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
01607                                     input->nsteps);
01608       if (input->nsteps)
01609         {
01610           gtk_toggle_button_set_active
01611             (GTK_TOGGLE_BUTTON (window->button_direction
01612                                 [input->direction]), TRUE);
01613           gtk_spin_button_set_value (window->spin_steps,
01614                                      (gdouble) input->nsteps);
01615           gtk_spin_button_set_value (window->spin_relaxation,
01616                                      (gdouble) input->relaxation);
01617           switch (input->direction)
01618             {
01619             case DIRECTION_METHOD_RANDOM:
01620               gtk_spin_button_set_value (window->spin_estimates,
01621                                          (gdouble) input->nestimates);
01622             }
01623         }
01624       break;
01625     default:
01626       gtk_spin_button_set_value (window->spin_population,
01627                                  (gdouble) input->nsimulations);
01628       gtk_spin_button_set_value (window->spin_generations,
01629                                  (gdouble) input->niterations);
01630       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01631       gtk_spin_button_set_value (window->spin_reproduction,
01632                                  input->reproduction_ratio);
01633       gtk_spin_button_set_value (window->spin_adaptation,
01634                                  input->adaptation_ratio);
01635     }
01636   gtk_toggle_button_set_active
01637     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01638   gtk_spin_button_set_value (window->spin_p, input->p);
01639   gtk_spin_button_set_value (window->spin_thresold, input->
      thresold);
01640   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01641   g_signal_handler_block (window->button_experiment,
01642                           window->id_experiment_name);
01643   gtk_combo_box_text_remove_all (window->combo_experiment);
01644   for (i = 0; i < input->nexperiments; ++i)
01645     gtk_combo_box_text_append_text (window->combo_experiment,
01646                                     input->experiment[i].name);
01647   g_signal_handler_unblock
01648     (window->button_experiment, window->
      id_experiment_name);
01649   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01650   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01651   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01652   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01653   gtk_combo_box_text_remove_all (window->combo_variable);
01654   for (i = 0; i < input->nvariables; ++i)
01655     gtk_combo_box_text_append_text (window->combo_variable,
01656                                     input->variable[i].name);
01657   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01658   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01659   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01660   window_set_variable ();
01661   window_update ();
01662
01663 #if DEBUG_INTERFACE
01664   fprintf (stderr, "window_read: end\n");
01665 #endif
01666   return 1;
```

```
01667 }
```

Here is the call graph for this function:

5.13.2.7 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 554 of file interface.c.

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

```
00618           input->algorithm = ALGORITHM_SWEEP;
00619           input->niterations
00620             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00621           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00622           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00623           window_save_direction ();
00624           break;
00625         default:
00626           input->algorithm = ALGORITHM_GENETIC;
00627           input->nsimulations
00628             = gtk_spin_button_get_value_as_int (window->spin_population);
00629           input->niterations
00630             = gtk_spin_button_get_value_as_int (window->spin_generations);
00631           input->mutation_ratio
00632             = gtk_spin_button_get_value (window->spin_mutation);
00633           input->reproduction_ratio
00634             = gtk_spin_button_get_value (window->spin_reproduction);
00635           input->adaptation_ratio
00636             = gtk_spin_button_get_value (window->spin_adaptation);
00637           break;
00638         }
00639       input->norm = window_get_norm ();
00640       input->p = gtk_spin_button_get_value (window->spin_p);
00641       input->thresold = gtk_spin_button_get_value (window->
    spin_thresold);
00642
00643       // Saving the XML file
00644       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00645       input_save (buffer);
00646
00647       // Closing and freeing memory
00648       g_free (buffer);
00649       gtk_widget_destroy (GTK_WIDGET (dlg));
00650 #if DEBUG_INTERFACE
00651       fprintf (stderr, "window_save: end\n");
00652 #endif
00653       return 1;
00654     }
00655
00656   // Closing and freeing memory
00657   gtk_widget_destroy (GTK_WIDGET (dlg));
00658 #if DEBUG_INTERFACE
00659   fprintf (stderr, "window_save: end\n");
00660 #endif
00661   return 0;
00662 }
```

Here is the call graph for this function:

5.13.2.8 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 1210 of file interface.c.

```
01211 {
01212   unsigned int i, j;
01213   char *buffer;
01214   GFile *file1, *file2;
01215 #if DEBUG_INTERFACE
01216   fprintf (stderr, "window_template_experiment: start\n");
01217 #endif
01218   i = (size_t) data;
01219   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220   file1
01221     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222   file2 = g_file_new_for_path (input->directory);
01223   buffer = g_file_get_relative_path (file2, file1);
01224   input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225   g_free (buffer);
01226   g_object_unref (file2);
01227   g_object_unref (file1);
01228 #if DEBUG_INTERFACE
01229   fprintf (stderr, "window_template_experiment: end\n");
01230 #endif
01231 }
```

5.14 interface.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
00049 {
00050   GtkDialog *dialog;
00051   GtkGrid *grid;
00052   GtkLabel *label_seed;
00054   GtkSpinButton *spin_seed;
00056   GtkLabel *label_threads;
00057   GtkSpinButton *spin_threads;
00058   GtkLabel *label_direction;
00059   GtkSpinButton *spin_direction;
00060 } Options;
00061
00066 typedef struct
00067 {
00068   GtkDialog *dialog;
00069   GtkLabel *label;
00070   GtkSpinner *spinner;
00071   GtkGrid *grid;
00072 } Running;
00073
00078 typedef struct
00079 {
00080   GtkWindow *window;
00081   GtkGrid *grid;
00082   GtkToolbar *bar_buttons;
00083   GtkToolButton *button_open;
00084   GtkToolButton *button_save;
00085   GtkToolButton *button_run;
00086   GtkToolButton *button_options;
00087   GtkToolButton *button_help;
00088   GtkToolButton *button_about;
00089   GtkToolButton *button_exit;
00090   GtkGrid *grid_files;
00091   GtkLabel *label_simulator;
00092   GtkFileChooserButton *button_simulator;
00094   GtkCheckButton *check_evaluator;
00095   GtkFileChooserButton *button_evaluator;
00097   GtkLabel *label_result;
00098   GtkEntry *entry_result;
00099   GtkLabel *label_variables;
00100   GtkEntry *entry_variables;
00101   GtkFrame *frame_norm;
00102   GtkGrid *grid_norm;
00103   GtkRadioButton *button_norm[NNORMS];
00105   GtkLabel *label_p;
00106   GtkSpinButton *spin_p;
00107   GtkScrolledWindow *scrolled_p;
00109   GtkFrame *frame_algorithm;
```

```
00110   GtkGrid *grid_algorithm;
00111   GtkRadioButton *button_algorithm[NALGORITHMS];
00113   GtkLabel *label_simulations;
00114   GtkSpinButton *spin_simulations;
00116   GtkLabel *label_iterations;
00117   GtkSpinButton *spin_iterations;
00119   GtkLabel *label_tolerance;
00120   GtkSpinButton *spin_tolerance;
00121   GtkLabel *label_bests;
00122   GtkSpinButton *spin_bests;
00123   GtkLabel *label_population;
00124   GtkSpinButton *spin_population;
00126   GtkLabel *label_generations;
00127   GtkSpinButton *spin_generations;
00129   GtkLabel *label_mutation;
00130   GtkSpinButton *spin_mutation;
00131   GtkLabel *label_reproduction;
00132   GtkSpinButton *spin_reproduction;
00134   GtkLabel *label_adaptation;
00135   GtkSpinButton *spin_adaptation;
00137   GtkCheckButton *check_direction;
00139   GtkGrid *grid_direction;
00141   GtkRadioButton *button_direction[NDIRECTIONS];
00143   GtkLabel *label_steps;
00144   GtkSpinButton *spin_steps;
00145   GtkLabel *label_estimates;
00146   GtkSpinButton *spin_estimates;
00148   GtkLabel *label_relaxation;
00150   GtkSpinButton *spin_relaxation;
00152   GtkLabel *label_thresold;
00153   GtkSpinButton *spin_thresold;
00154   GtkScrolledWindow *scrolled_thresold;
00156   GtkFrame *frame_variable;
00157   GtkGrid *grid_variable;
00158   GtkComboBoxText *combo_variable;
00160   GtkButton *button_add_variable;
00161   GtkButton *button_remove_variable;
00162   GtkLabel *label_variable;
00163   GtkEntry *entry_variable;
00164   GtkLabel *label_min;
00165   GtkSpinButton *spin_min;
00166   GtkScrolledWindow *scrolled_min;
00167   GtkLabel *label_max;
00168   GtkSpinButton *spin_max;
00169   GtkScrolledWindow *scrolled_max;
00170   GtkCheckButton *check_minabs;
00171   GtkSpinButton *spin_minabs;
00172   GtkScrolledWindow *scrolled_minabs;
00173   GtkCheckButton *check_maxabs;
00174   GtkSpinButton *spin_maxabs;
00175   GtkScrolledWindow *scrolled_maxabs;
00176   GtkLabel *label_precision;
00177   GtkSpinButton *spin_precision;
00178   GtkLabel *label_sweeps;
00179   GtkSpinButton *spin_sweeps;
00180   GtkLabel *label_bits;
00181   GtkSpinButton *spin_bits;
00182   GtkLabel *label_step;
00183   GtkSpinButton *spin_step;
00184   GtkScrolledWindow *scrolled_step;
00185   GtkFrame *frame_experiment;
00186   GtkGrid *grid_experiment;
00187   GtkComboBoxText *combo_experiment;
00188   GtkButton *button_add_experiment;
00189   GtkButton *button_remove_experiment;
00190   GtkLabel *label_experiment;
00191   GtkFileChooserButton *button_experiment;
00193   GtkLabel *label_weight;
00194   GtkSpinButton *spin_weight;
00195   GtkCheckButton *check_template[MAX_NINPUTS];
00197   GtkFileChooserButton *button_template[MAX_NINPUTS];
00199   GdkPixbuf *logo;
00200   Experiment *experiment;
00201   Variable *variable;
00202   char *application_directory;
00203   gulong id_experiment;
00204   gulong id_experiment_name;
00205   gulong id_variable;
00206   gulong id_variable_label;
00207   gulong id_template[MAX_NINPUTS];
00209   gulong id_input[MAX_NINPUTS];
00211   unsigned int nexperiments;
00212   unsigned int nvariables;
00213 } Window;
00214
00215 // Global variables
00216 extern const char *logo[];
```

```
00217 extern Options options[1];
00218 extern Running running[1];
00219 extern Window window[1];
00220
00221 // Public functions
00222 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00223 void input_save (char *filename);
00224 void options_new ();
00225 void running_new ();
00226 unsigned int window_get_algorithm ();
00227 unsigned int window_get_direction ();
00228 unsigned int window_get_norm ();
00229 void window_save_direction ();
00230 int window_save ();
00231 void window_run ();
00232 void window_help ();
00233 void window_update_direction ();
00234 void window_update ();
00235 void window_set_algorithm ();
00236 void window_set_experiment ();
00237 void window_remove_experiment ();
00238 void window_add_experiment ();
00239 void window_name_experiment ();
00240 void window_weight_experiment ();
00241 void window_inputs_experiment ();
00242 void window_template_experiment (void *data);
00243 void window_set_variable ();
00244 void window_remove_variable ();
00245 void window_add_variable ();
00246 void window_label_variable ();
00247 void window_precision_variable ();
00248 void window_rangemin_variable ();
00249 void window_rangemax_variable ();
00250 void window_rangeminabs_variable ();
00251 void window_rangemaxabs_variable ();
00252 void window_update_variable ();
00253 int window_read (char *filename);
00254 void window_open ();
00255 void window_new ();
00256
00257 #endif
```

5.15 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 "experiment.h"
#include "variable.h"
#include "input.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)

5.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file main.c.

5.16 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <glib/gstdio.h>
```

```
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #elif !defined (BSD)
00053 #include <alloca.h>
00054 #endif
00055 #if HAVE_MPI
00056 #include <mpi.h>
00057 #endif
00058 #if HAVE_GTK
00059 #include <gio/gio.h>
00060 #include <gtk/gtk.h>
00061 #endif
00062 #include "genetic/genetic.h"
00063 #include "utils.h"
00064 #include "experiment.h"
00065 #include "variable.h"
00066 #include "input.h"
00067 #include "optimize.h"
00068 #if HAVE_GTK
00069 #include "interface.h"
00070 #endif
00071
00072 #define DEBUG 0
00073
00074
00083 int
00084 main (int argn, char **argc)
00085 {
00086 #if HAVE_GTK
00087   char *buffer;
00088 #endif
00089
00090   // Starting pseudo-random numbers generator
00091 #if DEBUG
00092   fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00093 #endif
00094   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00095
00096   // Allowing spaces in the XML data file
00097 #if DEBUG
00098   fprintf (stderr, "main: allowing spaces in the XML data file\n");
00099 #endif
00100   xmlKeepBlanksDefault (0);
00101
00102   // Starting MPI
00103 #if HAVE_MPI
00104 #if DEBUG
00105   fprintf (stderr, "main: starting MPI\n");
00106 #endif
00107   MPI_Init (&argn, &argc);
00108   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00109   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00110   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00111 #else
00112   ntasks = 1;
00113 #endif
00114
00115   // Resetting result and variables file names
00116 #if DEBUG
00117   fprintf (stderr, "main: resetting result and variables file names\n");
00118 #endif
00119   input->result = input->variables = NULL;
00120
00121 #if HAVE_GTK
00122
00123   // Getting threads number and pseudo-random numbers generator seed
00124   nthreads_direction = nthreads = cores_number ();
00125   optimize->seed = DEFAULT_RANDOM_SEED;
00126
00127   // Setting local language and international floating point numbers notation
00128   setlocale (LC_ALL, "");
00129   setlocale (LC_NUMERIC, "C");
00130   window->application_directory = g_get_current_dir ();
00131   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
00132   bindtextdomain (PROGRAM_INTERFACE, buffer);
00133   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00134   textdomain (PROGRAM_INTERFACE);
00135
00136   // Initing GTK+
00137   gtk_disable_setlocale ();
00138   gtk_init (&argn, &argc);
00139
00140   // Opening the main window
00141   window_new ();
00142   gtk_main ();
00143
```

```
00144   // Freeing memory
00145   input_free ();
00146   g_free (buffer);
00147   gtk_widget_destroy (GTK_WIDGET (window->window));
00148   g_free (window->application_directory);
00149
00150 #else
00151
00152   // Checking syntax
00153   if (argn < 2)
00154     {
00155       printf ("The syntax is:\n"
00156               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00157               "[variables_file]\n");
00158       return 1;
00159     }
00160
00161   // Getting threads number and pseudo-random numbers generator seed
00162 #if DEBUG
00163   fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00164            "generator seed\n");
00165 #endif
00166   nthreads_direction = nthreads = cores_number ();
00167   optimize->seed = DEFAULT_RANDOM_SEED;
00168   if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00169     {
00170       nthreads_direction = nthreads = atoi (argc[2]);
00171       if (!nthreads)
00172         {
00173           printf ("Bad threads number\n");
00174           return 2;
00175         }
00176       argc += 2;
00177       argn -= 2;
00178       if (argn > 2 && !strcmp (argc[1], "-seed"))
00179         {
00180           optimize->seed = atoi (argc[2]);
00181           argc += 2;
00182           argn -= 2;
00183         }
00184     }
00185   else if (argn > 2 && !strcmp (argc[1], "-seed"))
00186     {
00187       optimize->seed = atoi (argc[2]);
00188       argc += 2;
00189       argn -= 2;
00190       if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00191         {
00192           nthreads_direction = nthreads = atoi (argc[2]);
00193           if (!nthreads)
00194             {
00195               printf ("Bad threads number\n");
00196               return 2;
00197             }
00198           argc += 2;
00199           argn -= 2;
00200         }
00201     }
00202   printf ("nthreads=%u\n", nthreads);
00203   printf ("seed=%lu\n", optimize->seed);
00204
00205   // Checking arguments
00206 #if DEBUG
00207   fprintf (stderr, "main: checking arguments\n");
00208 #endif
00209   if (argn > 4 || argn < 2)
00210     {
00211       printf ("The syntax is:\n"
00212               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00213               "[variables_file]\n");
00214       return 1;
00215     }
00216   if (argn > 2)
00217     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00218   if (argn == 4)
00219     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00220
00221   // Making optimization
00222 #if DEBUG
00223   fprintf (stderr, "main: making optimization\n");
00224 #endif
00225   if (input_open (argc[1]))
00226     optimize_open ();
00227
00228   // Freeing memory
00229 #if DEBUG
00230   fprintf (stderr, "main: freeing memory and closing\n");
```

```
00231 #endif
00232   optimize_free ();
00233
00234 #endif
00235
00236   // Closing MPI
00237 #if HAVE_MPI
00238   MPI_Finalize ();
00239 #endif
00240
00241   // Freeing memory
00242   gsl_rng_free (optimize->rng);
00243
00244   // Closing
00245   return 0;
00246 }
```

5.17 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 "experiment.h"
#include "variable.h"
#include "input.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 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.
- Optimize optimize [1]

  Optimization data.

5.17.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file optimize.c.

5.17.2 Function Documentation

5.17.2.1 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 461 of file optimize.c.

```
00462 {
00463   unsigned int i, j;
00464   double e;
00465 #if DEBUG
00466   fprintf (stderr, "optimize_best: start\n");
00467   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468            optimize->nsaveds, optimize->nbest);
00469 #endif
00470   if (optimize->nsaveds < optimize->nbest
00471       || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473       if (optimize->nsaveds < optimize->nbest)
00474         ++optimize->nsaveds;
```

```
00475       optimize->error_best[optimize->nsaveds - 1] = value;
00476       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477       for (i = optimize->nsaveds; --i;)
00478         {
00479           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00480             {
00481               j = optimize->simulation_best[i];
00482               e = optimize->error_best[i];
00483               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00484               optimize->error_best[i] = optimize->
      error_best[i - 1];
00485               optimize->simulation_best[i - 1] = j;
00486               optimize->error_best[i - 1] = e;
00487             }
00488           else
00489             break;
00490         }
00491     }
00492 #if DEBUG
00493   fprintf (stderr, "optimize_best: end\n");
00494 #endif
00495 }
```

5.17.2.2 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 786 of file optimize.c.

```
00787 {
00788 #if DEBUG
00789   fprintf (stderr, "optimize_best_direction: start\n");
00790   fprintf (stderr,
00791            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00792            simulation, value, optimize->error_best[0]);
00793 #endif
00794   if (value < optimize->error_best[0])
00795     {
00796       optimize->error_best[0] = value;
00797       optimize->simulation_best[0] = simulation;
00798 #if DEBUG
00799       fprintf (stderr,
00800                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00801                simulation, value);
00802 #endif
00803     }
00804 #if DEBUG
00805   fprintf (stderr, "optimize_best_direction: end\n");
00806 #endif
00807 }
```

5.17.2.3 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

simulation	Simulation number.

Definition at line 816 of file optimize.c.

```
00817 {
00818   unsigned int i, j;
00819   double e;
00820 #if DEBUG
00821   fprintf (stderr, "optimize_direction_sequential: start\n");
00822   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
```

```
00823            "nend_direction=%u\n",
00824            optimize->nstart_direction, optimize->
     nend_direction);
00825 #endif
00826   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00827     {
00828       j = simulation + i;
00829       e = optimize_norm (j);
00830       optimize_best_direction (j, e);
00831       optimize_save_variables (j, e);
00832       if (e < optimize->thresold)
00833         {
00834           optimize->stop = 1;
00835           break;
00836         }
00837 #if DEBUG
00838       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00839 #endif
00840     }
00841 #if DEBUG
00842   fprintf (stderr, "optimize_direction_sequential: end\n");
00843 #endif
00844 }
```

Here is the call graph for this function:

5.17.2.4 void * optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 854 of file optimize.c.

```
00855 {
00856   unsigned int i, thread;
00857   double e;
00858 #if DEBUG
00859   fprintf (stderr, "optimize_direction_thread: start\n");
00860 #endif
00861   thread = data->thread;
00862 #if DEBUG
00863   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864            thread,
00865            optimize->thread_direction[thread],
00866            optimize->thread_direction[thread + 1]);
00867 #endif
00868   for (i = optimize->thread_direction[thread];
00869        i < optimize->thread_direction[thread + 1]; ++i)
00870     {
00871       e = optimize_norm (i);
00872       g_mutex_lock (mutex);
00873       optimize_best_direction (i, e);
00874       optimize_save_variables (i, e);
00875       if (e < optimize->thresold)
00876         optimize->stop = 1;
00877       g_mutex_unlock (mutex);
00878       if (optimize->stop)
00879         break;
00880 #if DEBUG
00881       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882 #endif
00883     }
00884 #if DEBUG
00885   fprintf (stderr, "optimize_direction_thread: end\n");
00886 #endif
00887   g_thread_exit (NULL);
00888   return NULL;
00889 }
```

Here is the call graph for this function:

5.17.2.5 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 928 of file optimize.c.

```
00930 {
00931   double x;
00932 #if DEBUG
00933   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934 #endif
00935   x = optimize->direction[variable];
00936   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938       if (estimate & 1)
00939         x += optimize->step[variable];
00940       else
00941         x -= optimize->step[variable];
00942     }
00943 #if DEBUG
00944   fprintf (stderr,
00945            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946            variable, x);
00947   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00948 #endif
00949   return x;
00950 }
```

5.17.2.6 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 901 of file optimize.c.

```
00903 {
00904   double x;
00905 #if DEBUG
00906   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907 #endif
00908   x = optimize->direction[variable]
00909     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00910 #if DEBUG
00911   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00912            variable, x);
00913   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00914 #endif
00915   return x;
00916 }
```

5.17.2.7 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 1095 of file optimize.c.

```
01096 {
01097   unsigned int j;
01098   double objective;
01099   char buffer[64];
01100 #if DEBUG
01101   fprintf (stderr, "optimize_genetic_objective: start\n");
01102 #endif
01103   for (j = 0; j < optimize->nvariables; ++j)
01104     {
01105       optimize->value[entity->id * optimize->nvariables + j]
01106         = genetic_get_variable (entity, optimize->genetic_variable + j);
01107     }
01108   objective = optimize_norm (entity->id);
01109   g_mutex_lock (mutex);
01110   for (j = 0; j < optimize->nvariables; ++j)
01111     {
01112       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113       fprintf (optimize->file_variables, buffer,
01114                genetic_get_variable (entity, optimize->genetic_variable + j));
01115     }
01116   fprintf (optimize->file_variables, "%.14le\n", objective);
01117   g_mutex_unlock (mutex);
01118 #if DEBUG
01119   fprintf (stderr, "optimize_genetic_objective: end\n");
01120 #endif
01121   return objective;
01122 }
```

5.17.2.8 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 102 of file optimize.c.

```
00103 {
00104   unsigned int i;
00105   char buffer[32], value[32], *buffer2, *buffer3, *content;
00106   FILE *file;
00107   gsize length;
00108   GRegex *regex;
00109
00110 #if DEBUG
00111   fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114   // Checking the file
00115   if (!template)
00116     goto optimize_input_end;
00117
00118   // Opening template
00119   content = g_mapped_file_get_contents (template);
00120   length = g_mapped_file_get_length (template);
00121 #if DEBUG
00122   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124   file = g_fopen (input, "w");
00125
00126   // Parsing template
00127   for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG
00130       fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132       snprintf (buffer, 32, "@variable%u@", i + 1);
00133       regex = g_regex_new (buffer, 0, 0, NULL);
00134       if (i == 0)
00135         {
00136           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00137                                              optimize->label[i], 0, NULL);
00138 #if DEBUG
00139           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00140 #endif
00141         }
00142       else
```

```
00143         {
00144           length = strlen (buffer3);
00145           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00146                                              optimize->label[i], 0, NULL);
00147           g_free (buffer3);
00148         }
00149       g_regex_unref (regex);
00150       length = strlen (buffer2);
00151       snprintf (buffer, 32, "@value%u@", i + 1);
00152       regex = g_regex_new (buffer, 0, 0, NULL);
00153       snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->
      nvariables + i]);
00155 
00156 #if DEBUG
00157       fprintf (stderr, "optimize_input: value=%s\n", value);
00158 #endif
00159       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                          0, NULL);
00161       g_free (buffer2);
00162       g_regex_unref (regex);
00163     }
00164 
00165   // Saving input file
00166   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167   g_free (buffer3);
00168   fclose (file);
00169 
00170 optimize_input_end:
00171 #if DEBUG
00172   fprintf (stderr, "optimize_input: end\n");
00173 #endif
00174   return;
00175 }
```

5.17.2.9 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 584 of file optimize.c.

```
00586 {
00587   unsigned int i, j, k, s[optimize->nbest];
00588   double e[optimize->nbest];
00589 #if DEBUG
00590   fprintf (stderr, "optimize_merge: start\n");
00591 #endif
00592   i = j = k = 0;
00593   do
00594     {
00595       if (i == optimize->nsaveds)
00596         {
00597           s[k] = simulation_best[j];
00598           e[k] = error_best[j];
00599           ++j;
00600           ++k;
00601           if (j == nsaveds)
00602             break;
00603         }
00604       else if (j == nsaveds)
00605         {
00606           s[k] = optimize->simulation_best[i];
00607           e[k] = optimize->error_best[i];
00608           ++i;
00609           ++k;
00610           if (i == optimize->nsaveds)
00611             break;
00612         }
00613       else if (optimize->error_best[i] > error_best[j])
00614         {
00615           s[k] = simulation_best[j];
00616           e[k] = error_best[j];
00617           ++j;
00618           ++k;
```

```
00619         }
00620       else
00621         {
00622           s[k] = optimize->simulation_best[i];
00623           e[k] = optimize->error_best[i];
00624           ++i;
00625           ++k;
00626         }
00627     }
00628   while (k < optimize->nbest);
00629   optimize->nsaveds = k;
00630   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631   memcpy (optimize->error_best, e, k * sizeof (double));
00632 #if DEBUG
00633   fprintf (stderr, "optimize_merge: end\n");
00634 #endif
00635 }
```

5.17.2.10 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 294 of file optimize.c.

```
00295 {
00296   double e, ei;
00297   unsigned int i;
00298 #if DEBUG
00299   fprintf (stderr, "optimize_norm_euclidian: start\n");
00300 #endif
00301   e = 0.;
00302   for (i = 0; i < optimize->nexperiments; ++i)
00303     {
00304       ei = optimize_parse (simulation, i);
00305       e += ei * ei;
00306     }
00307   e = sqrt (e);
00308 #if DEBUG
00309   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310   fprintf (stderr, "optimize_norm_euclidian: end\n");
00311 #endif
00312   return e;
00313 }
```

Here is the call graph for this function:

5.17.2.11 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 323 of file optimize.c.

```
00324 {
00325   double e, ei;
00326   unsigned int i;
00327 #if DEBUG
00328   fprintf (stderr, "optimize_norm_maximum: start\n");
00329 #endif
00330   e = 0.;
00331   for (i = 0; i < optimize->nexperiments; ++i)
00332     {
00333       ei = fabs (optimize_parse (simulation, i));
00334       e = fmax (e, ei);
00335     }
00336 #if DEBUG
00337   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338   fprintf (stderr, "optimize_norm_maximum: end\n");
00339 #endif
00340   return e;
00341 }
```

Here is the call graph for this function:

5.17.2.12 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 351 of file optimize.c.

```
00352 {
00353   double e, ei;
00354   unsigned int i;
00355 #if DEBUG
00356   fprintf (stderr, "optimize_norm_p: start\n");
00357 #endif
00358   e = 0.;
00359   for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361       ei = fabs (optimize_parse (simulation, i));
00362       e += pow (ei, optimize->p);
00363     }
00364   e = pow (e, 1. / optimize->p);
00365 #if DEBUG
00366   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00367   fprintf (stderr, "optimize_norm_p: end\n");
00368 #endif
00369   return e;
00370 }
```

Here is the call graph for this function:

5.17.2.13 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 380 of file optimize.c.

```
00381 {
00382   double e;
00383   unsigned int i;
00384 #if DEBUG
00385   fprintf (stderr, "optimize_norm_taxicab: start\n");
00386 #endif
00387   e = 0.;
00388   for (i = 0; i < optimize->nexperiments; ++i)
00389     e += fabs (optimize_parse (simulation, i));
00390 #if DEBUG
00391   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392   fprintf (stderr, "optimize_norm_taxicab: end\n");
00393 #endif
00394   return e;
00395 }
```

Here is the call graph for this function:

5.17.2.14 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 188 of file optimize.c.

```
00189 {
00190   unsigned int i;
00191   double e;
00192   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00193     *buffer3, *buffer4;
00194   FILE *file_result;
00195
00196 #if DEBUG
00197   fprintf (stderr, "optimize_parse: start\n");
00198   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00199            experiment);
00200 #endif
00201
00202   // Opening input files
00203   for (i = 0; i < optimize->ninputs; ++i)
00204     {
00205       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00206 #if DEBUG
00207       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00208 #endif
00209       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00210     }
00211   for (; i < MAX_NINPUTS; ++i)
00212     strcpy (&input[i][0], "");
00213 #if DEBUG
00214   fprintf (stderr, "optimize_parse: parsing end\n");
00215 #endif
00216
00217   // Performing the simulation
00218   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00219   buffer2 = g_path_get_dirname (optimize->simulator);
00220   buffer3 = g_path_get_basename (optimize->simulator);
00221   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00222   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00223             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00224             input[6], input[7], output);
00225   g_free (buffer4);
00226   g_free (buffer3);
00227   g_free (buffer2);
00228 #if DEBUG
00229   fprintf (stderr, "optimize_parse: %s\n", buffer);
00230 #endif
00231   system (buffer);
```

```
00232
00233   // Checking the objective value function
00234   if (optimize->evaluator)
00235     {
00236       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00237       buffer2 = g_path_get_dirname (optimize->evaluator);
00238       buffer3 = g_path_get_basename (optimize->evaluator);
00239       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00240       snprintf (buffer, 512, "\"%s\" %s %s %s",
00241                 buffer4, output, optimize->experiment[experiment], result);
00242       g_free (buffer4);
00243       g_free (buffer3);
00244       g_free (buffer2);
00245 #if DEBUG
00246       fprintf (stderr, "optimize_parse: %s\n", buffer);
00247 #endif
00248       system (buffer);
00249       file_result = g_fopen (result, "r");
00250       e = atof (fgets (buffer, 512, file_result));
00251       fclose (file_result);
00252     }
00253   else
00254     {
00255       strcpy (result, "");
00256       file_result = g_fopen (output, "r");
00257       e = atof (fgets (buffer, 512, file_result));
00258       fclose (file_result);
00259     }
00260
00261   // Removing files
00262 #if !DEBUG
00263   for (i = 0; i < optimize->ninputs; ++i)
00264     {
00265       if (optimize->file[i][0])
00266         {
00267           snprintf (buffer, 512, RM " %s", &input[i][0]);
00268           system (buffer);
00269         }
00270     }
00271   snprintf (buffer, 512, RM " %s %s", output, result);
00272   system (buffer);
00273 #endif
00274
00275   // Processing pending events
00276   show_pending ();
00277
00278 #if DEBUG
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
```

Here is the call graph for this function:

5.17.2.15 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 433 of file optimize.c.

```
00434 {
00435   unsigned int i;
00436   char buffer[64];
00437 #if DEBUG
00438   fprintf (stderr, "optimize_save_variables: start\n");
00439 #endif
00440   for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00443       fprintf (optimize->file_variables, buffer,
00444                optimize->value[simulation * optimize->
      nvariables + i]);
00445     }
```

```
00446   fprintf (optimize->file_variables, "%.14le\n", error);
00447 #if DEBUG
00448   fprintf (stderr, "optimize_save_variables: end\n");
00449 #endif
00450 }
```

5.17.2.16 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 959 of file optimize.c.

```
00960 {
00961   GThread *thread[nthreads_direction];
00962   ParallelData data[nthreads_direction];
00963   unsigned int i, j, k, b;
00964 #if DEBUG
00965   fprintf (stderr, "optimize_step_direction: start\n");
00966 #endif
00967   for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969       k = (simulation + i) * optimize->nvariables;
00970       b = optimize->simulation_best[0] * optimize->
      nvariables;
00971 #if DEBUG
00972       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00973                simulation + i, optimize->simulation_best[0]);
00974 #endif
00975       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00976         {
00977 #if DEBUG
00978           fprintf (stderr,
00979                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00980                    i, j, optimize->value[b]);
00981 #endif
00982           optimize->value[k]
00983             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00984           optimize->value[k] = fmin (fmax (optimize->value[k],
00985                                            optimize->rangeminabs[j]),
00986                                      optimize->rangemaxabs[j]);
00987 #if DEBUG
00988           fprintf (stderr,
00989                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00990                    i, j, optimize->value[k]);
00991 #endif
00992         }
00993     }
00994   if (nthreads_direction == 1)
00995     optimize_direction_sequential (simulation);
00996   else
00997     {
00998       for (i = 0; i <= nthreads_direction; ++i)
00999         {
01000           optimize->thread_direction[i]
01001             = simulation + optimize->nstart_direction
01002             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01003             / nthreads_direction;
01004 #if DEBUG
01005           fprintf (stderr,
01006                    "optimize_step_direction: i=%u thread_direction=%u\n",
01007                    i, optimize->thread_direction[i]);
01008 #endif
01009         }
01010       for (i = 0; i < nthreads_direction; ++i)
01011         {
01012           data[i].thread = i;
01013           thread[i] = g_thread_new
01014             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01015         }
01016       for (i = 0; i < nthreads_direction; ++i)
01017         g_thread_join (thread[i]);
01018     }
01019 #if DEBUG
01020   fprintf (stderr, "optimize_step_direction: end\n");
01021 #endif
01022 }
```

Here is the call graph for this function:

5.17.2.17 void * optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 538 of file optimize.c.

```
00539 {
00540   unsigned int i, thread;
00541   double e;
00542 #if DEBUG
00543   fprintf (stderr, "optimize_thread: start\n");
00544 #endif
00545   thread = data->thread;
00546 #if DEBUG
00547   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00548            optimize->thread[thread], optimize->thread[thread + 1]);
00549 #endif
00550   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552       e = optimize_norm (i);
00553       g_mutex_lock (mutex);
00554       optimize_best (i, e);
00555       optimize_save_variables (i, e);
00556       if (e < optimize->thresold)
00557         optimize->stop = 1;
00558       g_mutex_unlock (mutex);
00559       if (optimize->stop)
00560         break;
00561 #if DEBUG
00562       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00563 #endif
00564     }
00565 #if DEBUG
00566   fprintf (stderr, "optimize_thread: end\n");
00567 #endif
00568   g_thread_exit (NULL);
00569   return NULL;
00570 }
```

Here is the call graph for this function:

5.18 optimize.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elif !defined (BSD)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "utils.h"
00059 #include "experiment.h"
00060 #include "variable.h"
00061 #include "input.h"
00062 #include "optimize.h"
00063
00064 #define DEBUG 0
00065
00066
00070 #ifdef G_OS_WIN32
00071 #define RM "del"
00072 #else
00073 #define RM "rm"
00074 #endif
00075
00076 int ntasks;
00077 unsigned int nthreads;
00078 unsigned int nthreads_direction;
00080 GMutex mutex[1];
00081 void (*optimize_algorithm) ();
00083 double (*optimize_estimate_direction) (unsigned int variable,
00084                                        unsigned int estimate);
00086 double (*optimize_norm) (unsigned int simulation);
00088 Optimize optimize[1];
00089
00101 void
00102 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00103 {
00104   unsigned int i;
00105   char buffer[32], value[32], *buffer2, *buffer3, *content;
00106   FILE *file;
00107   gsize length;
00108   GRegex *regex;
00109
00110 #if DEBUG
00111   fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114   // Checking the file
00115   if (!template)
00116     goto optimize_input_end;
00117
00118   // Opening template
00119   content = g_mapped_file_get_contents (template);
00120   length = g_mapped_file_get_length (template);
00121 #if DEBUG
00122   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124   file = g_fopen (input, "w");
00125
00126   // Parsing template
00127   for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG
00130       fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132       snprintf (buffer, 32, "@variable%u@", i + 1);
```

```
00133       regex = g_regex_new (buffer, 0, 0, NULL);
00134       if (i == 0)
00135         {
00136           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00137                                              optimize->label[i], 0, NULL);
00138 #if DEBUG
00139           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00140 #endif
00141         }
00142       else
00143         {
00144           length = strlen (buffer3);
00145           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00146                                              optimize->label[i], 0, NULL);
00147           g_free (buffer3);
00148         }
00149       g_regex_unref (regex);
00150       length = strlen (buffer2);
00151       snprintf (buffer, 32, "@value%u@", i + 1);
00152       regex = g_regex_new (buffer, 0, 0, NULL);
00153       snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->nvariables + i]);
00155
00156 #if DEBUG
00157       fprintf (stderr, "optimize_input: value=%s\n", value);
00158 #endif
00159       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                          0, NULL);
00161       g_free (buffer2);
00162       g_regex_unref (regex);
00163     }
00164
00165   // Saving input file
00166   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167   g_free (buffer3);
00168   fclose (file);
00169
00170 optimize_input_end:
00171 #if DEBUG
00172   fprintf (stderr, "optimize_input: end\n");
00173 #endif
00174   return;
00175 }
00176
00187 double
00188 optimize_parse (unsigned int simulation, unsigned int experiment)
00189 {
00190   unsigned int i;
00191   double e;
00192   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00193     *buffer3, *buffer4;
00194   FILE *file_result;
00195
00196 #if DEBUG
00197   fprintf (stderr, "optimize_parse: start\n");
00198   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00199            experiment);
00200 #endif
00201
00202   // Opening input files
00203   for (i = 0; i < optimize->ninputs; ++i)
00204     {
00205       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00206 #if DEBUG
00207       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00208 #endif
00209       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00210     }
00211   for (; i < MAX_NINPUTS; ++i)
00212     strcpy (&input[i][0], "");
00213 #if DEBUG
00214   fprintf (stderr, "optimize_parse: parsing end\n");
00215 #endif
00216
00217   // Performing the simulation
00218   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00219   buffer2 = g_path_get_dirname (optimize->simulator);
00220   buffer3 = g_path_get_basename (optimize->simulator);
00221   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00222   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00223             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00224             input[6], input[7], output);
00225   g_free (buffer4);
00226   g_free (buffer3);
00227   g_free (buffer2);
00228 #if DEBUG
00229   fprintf (stderr, "optimize_parse: %s\n", buffer);
```

```
00230 #endif
00231   system (buffer);
00232
00233   // Checking the objective value function
00234   if (optimize->evaluator)
00235     {
00236       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00237       buffer2 = g_path_get_dirname (optimize->evaluator);
00238       buffer3 = g_path_get_basename (optimize->evaluator);
00239       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00240       snprintf (buffer, 512, "\"%s\" %s %s %s",
00241                 buffer4, output, optimize->experiment[experiment], result);
00242       g_free (buffer4);
00243       g_free (buffer3);
00244       g_free (buffer2);
00245 #if DEBUG
00246       fprintf (stderr, "optimize_parse: %s\n", buffer);
00247 #endif
00248       system (buffer);
00249       file_result = g_fopen (result, "r");
00250       e = atof (fgets (buffer, 512, file_result));
00251       fclose (file_result);
00252     }
00253   else
00254     {
00255       strcpy (result, "");
00256       file_result = g_fopen (output, "r");
00257       e = atof (fgets (buffer, 512, file_result));
00258       fclose (file_result);
00259     }
00260
00261   // Removing files
00262 #if !DEBUG
00263   for (i = 0; i < optimize->ninputs; ++i)
00264     {
00265       if (optimize->file[i][0])
00266         {
00267           snprintf (buffer, 512, RM " %s", &input[i][0]);
00268           system (buffer);
00269         }
00270     }
00271   snprintf (buffer, 512, RM " %s %s", output, result);
00272   system (buffer);
00273 #endif
00274
00275   // Processing pending events
00276   show_pending ();
00277
00278 #if DEBUG
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
00285
00293 double
00294 optimize_norm_euclidian (unsigned int simulation)
00295 {
00296   double e, ei;
00297   unsigned int i;
00298 #if DEBUG
00299   fprintf (stderr, "optimize_norm_euclidian: start\n");
00300 #endif
00301   e = 0.;
00302   for (i = 0; i < optimize->nexperiments; ++i)
00303     {
00304       ei = optimize_parse (simulation, i);
00305       e += ei * ei;
00306     }
00307   e = sqrt (e);
00308 #if DEBUG
00309   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310   fprintf (stderr, "optimize_norm_euclidian: end\n");
00311 #endif
00312   return e;
00313 }
00314
00322 double
00323 optimize_norm_maximum (unsigned int simulation)
00324 {
00325   double e, ei;
00326   unsigned int i;
00327 #if DEBUG
00328   fprintf (stderr, "optimize_norm_maximum: start\n");
00329 #endif
00330   e = 0.;
```

```
00331   for (i = 0; i < optimize->nexperiments; ++i)
00332     {
00333       ei = fabs (optimize_parse (simulation, i));
00334       e = fmax (e, ei);
00335     }
00336 #if DEBUG
00337   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338   fprintf (stderr, "optimize_norm_maximum: end\n");
00339 #endif
00340   return e;
00341 }
00342
00350 double
00351 optimize_norm_p (unsigned int simulation)
00352 {
00353   double e, ei;
00354   unsigned int i;
00355 #if DEBUG
00356   fprintf (stderr, "optimize_norm_p: start\n");
00357 #endif
00358   e = 0.;
00359   for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361       ei = fabs (optimize_parse (simulation, i));
00362       e += pow (ei, optimize->p);
00363     }
00364   e = pow (e, 1. / optimize->p);
00365 #if DEBUG
00366   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00367   fprintf (stderr, "optimize_norm_p: end\n");
00368 #endif
00369   return e;
00370 }
00371
00379 double
00380 optimize_norm_taxicab (unsigned int simulation)
00381 {
00382   double e;
00383   unsigned int i;
00384 #if DEBUG
00385   fprintf (stderr, "optimize_norm_taxicab: start\n");
00386 #endif
00387   e = 0.;
00388   for (i = 0; i < optimize->nexperiments; ++i)
00389     e += fabs (optimize_parse (simulation, i));
00390 #if DEBUG
00391   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392   fprintf (stderr, "optimize_norm_taxicab: end\n");
00393 #endif
00394   return e;
00395 }
00396
00401 void
00402 optimize_print ()
00403 {
00404   unsigned int i;
00405   char buffer[512];
00406 #if HAVE_MPI
00407   if (optimize->mpi_rank)
00408     return;
00409 #endif
00410   printf ("%s\n", gettext ("Best result"));
00411   fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
00412   printf ("error = %.15le\n", optimize->error_old[0]);
00413   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
00414   for (i = 0; i < optimize->nvariables; ++i)
00415     {
00416       snprintf (buffer, 512, "%s = %s\n",
00417                 optimize->label[i], format[optimize->precision[i]]);
00418       printf (buffer, optimize->value_old[i]);
00419       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00420     }
00421   fflush (optimize->file_result);
00422 }
00423
00432 void
00433 optimize_save_variables (unsigned int simulation, double error)
00434 {
00435   unsigned int i;
00436   char buffer[64];
00437 #if DEBUG
00438   fprintf (stderr, "optimize_save_variables: start\n");
00439 #endif
00440   for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
```

```
00443       fprintf (optimize->file_variables, buffer,
00444                optimize->value[simulation * optimize->nvariables + i]);
00445     }
00446   fprintf (optimize->file_variables, "%.14le\n", error);
00447 #if DEBUG
00448   fprintf (stderr, "optimize_save_variables: end\n");
00449 #endif
00450 }
00451
00460 void
00461 optimize_best (unsigned int simulation, double value)
00462 {
00463   unsigned int i, j;
00464   double e;
00465 #if DEBUG
00466   fprintf (stderr, "optimize_best: start\n");
00467   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468            optimize->nsaveds, optimize->nbest);
00469 #endif
00470   if (optimize->nsaveds < optimize->nbest
00471       || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473       if (optimize->nsaveds < optimize->nbest)
00474         ++optimize->nsaveds;
00475       optimize->error_best[optimize->nsaveds - 1] = value;
00476       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477       for (i = optimize->nsaveds; --i;)
00478         {
00479           if (optimize->error_best[i] < optimize->error_best[i - 1])
00480             {
00481               j = optimize->simulation_best[i];
00482               e = optimize->error_best[i];
00483               optimize->simulation_best[i] = optimize->
    simulation_best[i - 1];
00484               optimize->error_best[i] = optimize->error_best[i - 1];
00485               optimize->simulation_best[i - 1] = j;
00486               optimize->error_best[i - 1] = e;
00487             }
00488           else
00489             break;
00490         }
00491     }
00492 #if DEBUG
00493   fprintf (stderr, "optimize_best: end\n");
00494 #endif
00495 }
00496
00501 void
00502 optimize_sequential ()
00503 {
00504   unsigned int i;
00505   double e;
00506 #if DEBUG
00507   fprintf (stderr, "optimize_sequential: start\n");
00508   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00509            optimize->nstart, optimize->nend);
00510 #endif
00511   for (i = optimize->nstart; i < optimize->nend; ++i)
00512     {
00513       e = optimize_norm (i);
00514       optimize_best (i, e);
00515       optimize_save_variables (i, e);
00516       if (e < optimize->thresold)
00517         {
00518           optimize->stop = 1;
00519           break;
00520         }
00521 #if DEBUG
00522       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00523 #endif
00524     }
00525 #if DEBUG
00526   fprintf (stderr, "optimize_sequential: end\n");
00527 #endif
00528 }
00529
00537 void *
00538 optimize_thread (ParallelData * data)
00539 {
00540   unsigned int i, thread;
00541   double e;
00542 #if DEBUG
00543   fprintf (stderr, "optimize_thread: start\n");
00544 #endif
00545   thread = data->thread;
00546 #if DEBUG
00547   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
```

```
00548           optimize->thread[thread], optimize->thread[thread + 1]);
00549 #endif
00550   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552       e = optimize_norm (i);
00553       g_mutex_lock (mutex);
00554       optimize_best (i, e);
00555       optimize_save_variables (i, e);
00556       if (e < optimize->thresold)
00557         optimize->stop = 1;
00558       g_mutex_unlock (mutex);
00559       if (optimize->stop)
00560         break;
00561 #if DEBUG
00562       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00563 #endif
00564     }
00565 #if DEBUG
00566   fprintf (stderr, "optimize_thread: end\n");
00567 #endif
00568   g_thread_exit (NULL);
00569   return NULL;
00570 }
00571
00583 void
00584 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00585                 double *error_best)
00586 {
00587   unsigned int i, j, k, s[optimize->nbest];
00588   double e[optimize->nbest];
00589 #if DEBUG
00590   fprintf (stderr, "optimize_merge: start\n");
00591 #endif
00592   i = j = k = 0;
00593   do
00594     {
00595       if (i == optimize->nsaveds)
00596         {
00597           s[k] = simulation_best[j];
00598           e[k] = error_best[j];
00599           ++j;
00600           ++k;
00601           if (j == nsaveds)
00602             break;
00603         }
00604       else if (j == nsaveds)
00605         {
00606           s[k] = optimize->simulation_best[i];
00607           e[k] = optimize->error_best[i];
00608           ++i;
00609           ++k;
00610           if (i == optimize->nsaveds)
00611             break;
00612         }
00613       else if (optimize->error_best[i] > error_best[j])
00614         {
00615           s[k] = simulation_best[j];
00616           e[k] = error_best[j];
00617           ++j;
00618           ++k;
00619         }
00620       else
00621         {
00622           s[k] = optimize->simulation_best[i];
00623           e[k] = optimize->error_best[i];
00624           ++i;
00625           ++k;
00626         }
00627     }
00628   while (k < optimize->nbest);
00629   optimize->nsaveds = k;
00630   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631   memcpy (optimize->error_best, e, k * sizeof (double));
00632 #if DEBUG
00633   fprintf (stderr, "optimize_merge: end\n");
00634 #endif
00635 }
00636
00641 #if HAVE_MPI
00642 void
00643 optimize_synchronise ()
00644 {
00645   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00646   double error_best[optimize->nbest];
00647   MPI_Status mpi_stat;
00648 #if DEBUG
00649   fprintf (stderr, "optimize_synchronise: start\n");
```

```
00650 #endif
00651   if (optimize->mpi_rank == 0)
00652     {
00653       for (i = 1; i < ntasks; ++i)
00654         {
00655           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00656           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00657                     MPI_COMM_WORLD, &mpi_stat);
00658           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00659                     MPI_COMM_WORLD, &mpi_stat);
00660           optimize_merge (nsaveds, simulation_best, error_best);
00661           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00662           if (stop)
00663             optimize->stop = 1;
00664         }
00665       for (i = 1; i < ntasks; ++i)
00666         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00667     }
00668   else
00669     {
00670       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00671       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00672                 MPI_COMM_WORLD);
00673       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00674                 MPI_COMM_WORLD);
00675       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00676       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00677       if (stop)
00678         optimize->stop = 1;
00679     }
00680 #if DEBUG
00681   fprintf (stderr, "optimize_synchronise: end\n");
00682 #endif
00683 }
00684 #endif
00685
00690 void
00691 optimize_sweep ()
00692 {
00693   unsigned int i, j, k, l;
00694   double e;
00695   GThread *thread[nthreads];
00696   ParallelData data[nthreads];
00697 #if DEBUG
00698   fprintf (stderr, "optimize_sweep: start\n");
00699 #endif
00700   for (i = 0; i < optimize->nsimulations; ++i)
00701     {
00702       k = i;
00703       for (j = 0; j < optimize->nvariables; ++j)
00704         {
00705           l = k % optimize->nsweeps[j];
00706           k /= optimize->nsweeps[j];
00707           e = optimize->rangemin[j];
00708           if (optimize->nsweeps[j] > 1)
00709             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00710               / (optimize->nsweeps[j] - 1);
00711           optimize->value[i * optimize->nvariables + j] = e;
00712         }
00713     }
00714   optimize->nsaveds = 0;
00715   if (nthreads <= 1)
00716     optimize_sequential ();
00717   else
00718     {
00719       for (i = 0; i < nthreads; ++i)
00720         {
00721           data[i].thread = i;
00722           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00723         }
00724       for (i = 0; i < nthreads; ++i)
00725         g_thread_join (thread[i]);
00726     }
00727 #if HAVE_MPI
00728   // Communicating tasks results
00729   optimize_synchronise ();
00730 #endif
00731 #if DEBUG
00732   fprintf (stderr, "optimize_sweep: end\n");
00733 #endif
00734 }
00735
00740 void
00741 optimize_MonteCarlo ()
00742 {
00743   unsigned int i, j;
00744   GThread *thread[nthreads];
```

```
00745   ParallelData data[nthreads];
00746 #if DEBUG
00747   fprintf (stderr, "optimize_MonteCarlo: start\n");
00748 #endif
00749   for (i = 0; i < optimize->nsimulations; ++i)
00750     for (j = 0; j < optimize->nvariables; ++j)
00751       optimize->value[i * optimize->nvariables + j]
00752         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00753         * (optimize->rangemax[j] - optimize->rangemin[j]);
00754   optimize->nsaveds = 0;
00755   if (nthreads <= 1)
00756     optimize_sequential ();
00757   else
00758     {
00759       for (i = 0; i < nthreads; ++i)
00760         {
00761           data[i].thread = i;
00762           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00763         }
00764       for (i = 0; i < nthreads; ++i)
00765         g_thread_join (thread[i]);
00766     }
00767 #if HAVE_MPI
00768   // Communicating tasks results
00769   optimize_synchronise ();
00770 #endif
00771 #if DEBUG
00772   fprintf (stderr, "optimize_MonteCarlo: end\n");
00773 #endif
00774 }
00775
00785 void
00786 optimize_best_direction (unsigned int simulation, double value)
00787 {
00788 #if DEBUG
00789   fprintf (stderr, "optimize_best_direction: start\n");
00790   fprintf (stderr,
00791            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00792            simulation, value, optimize->error_best[0]);
00793 #endif
00794   if (value < optimize->error_best[0])
00795     {
00796       optimize->error_best[0] = value;
00797       optimize->simulation_best[0] = simulation;
00798 #if DEBUG
00799       fprintf (stderr,
00800                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00801                simulation, value);
00802 #endif
00803     }
00804 #if DEBUG
00805   fprintf (stderr, "optimize_best_direction: end\n");
00806 #endif
00807 }
00808
00815 void
00816 optimize_direction_sequential (unsigned int simulation)
00817 {
00818   unsigned int i, j;
00819   double e;
00820 #if DEBUG
00821   fprintf (stderr, "optimize_direction_sequential: start\n");
00822   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00823            "nend_direction=%u\n",
00824            optimize->nstart_direction, optimize->nend_direction);
00825 #endif
00826   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00827     {
00828       j = simulation + i;
00829       e = optimize_norm (j);
00830       optimize_best_direction (j, e);
00831       optimize_save_variables (j, e);
00832       if (e < optimize->thresold)
00833         {
00834           optimize->stop = 1;
00835           break;
00836         }
00837 #if DEBUG
00838       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00839 #endif
00840     }
00841 #if DEBUG
00842   fprintf (stderr, "optimize_direction_sequential: end\n");
00843 #endif
00844 }
00845
00853 void *
```

```
00854 optimize_direction_thread (ParallelData * data)
00855 {
00856   unsigned int i, thread;
00857   double e;
00858 #if DEBUG
00859   fprintf (stderr, "optimize_direction_thread: start\n");
00860 #endif
00861   thread = data->thread;
00862 #if DEBUG
00863   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864            thread,
00865            optimize->thread_direction[thread],
00866            optimize->thread_direction[thread + 1]);
00867 #endif
00868   for (i = optimize->thread_direction[thread];
00869        i < optimize->thread_direction[thread + 1]; ++i)
00870     {
00871       e = optimize_norm (i);
00872       g_mutex_lock (mutex);
00873       optimize_best_direction (i, e);
00874       optimize_save_variables (i, e);
00875       if (e < optimize->thresold)
00876         optimize->stop = 1;
00877       g_mutex_unlock (mutex);
00878       if (optimize->stop)
00879         break;
00880 #if DEBUG
00881       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882 #endif
00883     }
00884 #if DEBUG
00885   fprintf (stderr, "optimize_direction_thread: end\n");
00886 #endif
00887   g_thread_exit (NULL);
00888   return NULL;
00889 }
00890
00900 double
00901 optimize_estimate_direction_random (unsigned int variable,
00902                                     unsigned int estimate)
00903 {
00904   double x;
00905 #if DEBUG
00906   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907 #endif
00908   x = optimize->direction[variable]
00909     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00910 #if DEBUG
00911   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00912            variable, x);
00913   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00914 #endif
00915   return x;
00916 }
00917
00927 double
00928 optimize_estimate_direction_coordinates (unsigned int variable,
00929                                          unsigned int estimate)
00930 {
00931   double x;
00932 #if DEBUG
00933   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934 #endif
00935   x = optimize->direction[variable];
00936   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938       if (estimate & 1)
00939         x += optimize->step[variable];
00940       else
00941         x -= optimize->step[variable];
00942     }
00943 #if DEBUG
00944   fprintf (stderr,
00945            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946            variable, x);
00947   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00948 #endif
00949   return x;
00950 }
00951
00958 void
00959 optimize_step_direction (unsigned int simulation)
00960 {
00961   GThread *thread[nthreads_direction];
00962   ParallelData data[nthreads_direction];
00963   unsigned int i, j, k, b;
00964 #if DEBUG
```

```
00965   fprintf (stderr, "optimize_step_direction: start\n");
00966 #endif
00967   for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969       k = (simulation + i) * optimize->nvariables;
00970       b = optimize->simulation_best[0] * optimize->nvariables;
00971 #if DEBUG
00972       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00973                simulation + i, optimize->simulation_best[0]);
00974 #endif
00975       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00976         {
00977 #if DEBUG
00978           fprintf (stderr,
00979                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00980                    i, j, optimize->value[b]);
00981 #endif
00982           optimize->value[k]
00983             = optimize->value[b] + optimize_estimate_direction (j, i);
00984           optimize->value[k] = fmin (fmax (optimize->value[k],
00985                                            optimize->rangeminabs[j]),
00986                                      optimize->rangemaxabs[j]);
00987 #if DEBUG
00988           fprintf (stderr,
00989                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00990                    i, j, optimize->value[k]);
00991 #endif
00992         }
00993     }
00994   if (nthreads_direction == 1)
00995     optimize_direction_sequential (simulation);
00996   else
00997     {
00998       for (i = 0; i <= nthreads_direction; ++i)
00999         {
01000           optimize->thread_direction[i]
01001             = simulation + optimize->nstart_direction
01002             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01003             / nthreads_direction;
01004 #if DEBUG
01005           fprintf (stderr,
01006                    "optimize_step_direction: i=%u thread_direction=%u\n",
01007                    i, optimize->thread_direction[i]);
01008 #endif
01009         }
01010       for (i = 0; i < nthreads_direction; ++i)
01011         {
01012           data[i].thread = i;
01013           thread[i] = g_thread_new
01014             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01015         }
01016       for (i = 0; i < nthreads_direction; ++i)
01017         g_thread_join (thread[i]);
01018     }
01019 #if DEBUG
01020   fprintf (stderr, "optimize_step_direction: end\n");
01021 #endif
01022 }
01023
01028 void
01029 optimize_direction ()
01030 {
01031   unsigned int i, j, k, b, s, adjust;
01032 #if DEBUG
01033   fprintf (stderr, "optimize_direction: start\n");
01034 #endif
01035   for (i = 0; i < optimize->nvariables; ++i)
01036     optimize->direction[i] = 0.;
01037   b = optimize->simulation_best[0] * optimize->nvariables;
01038   s = optimize->nsimulations;
01039   adjust = 1;
01040   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01041     {
01042 #if DEBUG
01043       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01044                i, optimize->simulation_best[0]);
01045 #endif
01046       optimize_step_direction (s);
01047       k = optimize->simulation_best[0] * optimize->nvariables;
01048 #if DEBUG
01049       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01050                i, optimize->simulation_best[0]);
01051 #endif
01052       if (k == b)
01053         {
01054           if (adjust)
```

```
01055             for (j = 0; j < optimize->nvariables; ++j)
01056               optimize->step[j] *= 0.5;
01057           for (j = 0; j < optimize->nvariables; ++j)
01058             optimize->direction[j] = 0.;
01059           adjust = 1;
01060         }
01061       else
01062         {
01063           for (j = 0; j < optimize->nvariables; ++j)
01064             {
01065 #if DEBUG
01066               fprintf (stderr,
01067                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01068                        j, optimize->value[k + j], j, optimize->value[b + j]);
01069 #endif
01070               optimize->direction[j]
01071                 = (1. - optimize->relaxation) * optimize->direction[j]
01072                 + optimize->relaxation
01073                 * (optimize->value[k + j] - optimize->value[b + j]);
01074 #if DEBUG
01075               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01076                        j, optimize->direction[j]);
01077 #endif
01078             }
01079           adjust = 0;
01080         }
01081     }
01082 #if DEBUG
01083   fprintf (stderr, "optimize_direction: end\n");
01084 #endif
01085 }
01086
01094 double
01095 optimize_genetic_objective (Entity * entity)
01096 {
01097   unsigned int j;
01098   double objective;
01099   char buffer[64];
01100 #if DEBUG
01101   fprintf (stderr, "optimize_genetic_objective: start\n");
01102 #endif
01103   for (j = 0; j < optimize->nvariables; ++j)
01104     {
01105       optimize->value[entity->id * optimize->nvariables + j]
01106         = genetic_get_variable (entity, optimize->genetic_variable + j);
01107     }
01108   objective = optimize_norm (entity->id);
01109   g_mutex_lock (mutex);
01110   for (j = 0; j < optimize->nvariables; ++j)
01111     {
01112       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113       fprintf (optimize->file_variables, buffer,
01114                genetic_get_variable (entity, optimize->genetic_variable + j));
01115     }
01116   fprintf (optimize->file_variables, "%.14le\n", objective);
01117   g_mutex_unlock (mutex);
01118 #if DEBUG
01119   fprintf (stderr, "optimize_genetic_objective: end\n");
01120 #endif
01121   return objective;
01122 }
01123
01128 void
01129 optimize_genetic ()
01130 {
01131   char *best_genome;
01132   double best_objective, *best_variable;
01133 #if DEBUG
01134   fprintf (stderr, "optimize_genetic: start\n");
01135   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01136            nthreads);
01137   fprintf (stderr,
01138            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01139            optimize->nvariables, optimize->nsimulations, optimize->
      niterations);
01140   fprintf (stderr,
01141            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01142            optimize->mutation_ratio, optimize->reproduction_ratio,
01143            optimize->adaptation_ratio);
01144 #endif
01145   genetic_algorithm_default (optimize->nvariables,
01146                              optimize->genetic_variable,
01147                              optimize->nsimulations,
01148                              optimize->niterations,
01149                              optimize->mutation_ratio,
01150                              optimize->reproduction_ratio,
01151                              optimize->adaptation_ratio,
```

```
01152                               optimize->seed,
01153                               optimize->thresold,
01154                               &optimize_genetic_objective,
01155                               &best_genome, &best_variable, &best_objective);
01156 #if DEBUG
01157   fprintf (stderr, "optimize_genetic: the best\n");
01158 #endif
01159   optimize->error_old = (double *) g_malloc (sizeof (double));
01160   optimize->value_old
01161     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01162   optimize->error_old[0] = best_objective;
01163   memcpy (optimize->value_old, best_variable,
01164           optimize->nvariables * sizeof (double));
01165   g_free (best_genome);
01166   g_free (best_variable);
01167   optimize_print ();
01168 #if DEBUG
01169   fprintf (stderr, "optimize_genetic: end\n");
01170 #endif
01171 }
01172
01177 void
01178 optimize_save_old ()
01179 {
01180   unsigned int i, j;
01181 #if DEBUG
01182   fprintf (stderr, "optimize_save_old: start\n");
01183   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01184 #endif
01185   memcpy (optimize->error_old, optimize->error_best,
01186           optimize->nbest * sizeof (double));
01187   for (i = 0; i < optimize->nbest; ++i)
01188     {
01189       j = optimize->simulation_best[i];
01190 #if DEBUG
01191       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01192 #endif
01193       memcpy (optimize->value_old + i * optimize->nvariables,
01194               optimize->value + j * optimize->nvariables,
01195               optimize->nvariables * sizeof (double));
01196     }
01197 #if DEBUG
01198   for (i = 0; i < optimize->nvariables; ++i)
01199     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01200              i, optimize->value_old[i]);
01201   fprintf (stderr, "optimize_save_old: end\n");
01202 #endif
01203 }
01204
01210 void
01211 optimize_merge_old ()
01212 {
01213   unsigned int i, j, k;
01214   double v[optimize->nbest * optimize->nvariables], e[optimize->
      nbest],
01215     *enew, *eold;
01216 #if DEBUG
01217   fprintf (stderr, "optimize_merge_old: start\n");
01218 #endif
01219   enew = optimize->error_best;
01220   eold = optimize->error_old;
01221   i = j = k = 0;
01222   do
01223     {
01224       if (*enew < *eold)
01225         {
01226           memcpy (v + k * optimize->nvariables,
01227                   optimize->value
01228                   + optimize->simulation_best[i] * optimize->
      nvariables,
01229                   optimize->nvariables * sizeof (double));
01230           e[k] = *enew;
01231           ++k;
01232           ++enew;
01233           ++i;
01234         }
01235       else
01236         {
01237           memcpy (v + k * optimize->nvariables,
01238                   optimize->value_old + j * optimize->nvariables,
01239                   optimize->nvariables * sizeof (double));
01240           e[k] = *eold;
01241           ++k;
01242           ++eold;
01243           ++j;
01244         }
01245     }
```

```
01246   while (k < optimize->nbest);
01247   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01248   memcpy (optimize->error_old, e, k * sizeof (double));
01249 #if DEBUG
01250   fprintf (stderr, "optimize_merge_old: end\n");
01251 #endif
01252 }
01253
01259 void
01260 optimize_refine ()
01261 {
01262   unsigned int i, j;
01263   double d;
01264 #if HAVE_MPI
01265   MPI_Status mpi_stat;
01266 #endif
01267 #if DEBUG
01268   fprintf (stderr, "optimize_refine: start\n");
01269 #endif
01270 #if HAVE_MPI
01271   if (!optimize->mpi_rank)
01272     {
01273 #endif
01274       for (j = 0; j < optimize->nvariables; ++j)
01275         {
01276           optimize->rangemin[j] = optimize->rangemax[j]
01277             = optimize->value_old[j];
01278         }
01279       for (i = 0; ++i < optimize->nbest;)
01280         {
01281           for (j = 0; j < optimize->nvariables; ++j)
01282             {
01283               optimize->rangemin[j]
01284                 = fmin (optimize->rangemin[j],
01285                         optimize->value_old[i * optimize->nvariables + j]);
01286               optimize->rangemax[j]
01287                 = fmax (optimize->rangemax[j],
01288                         optimize->value_old[i * optimize->nvariables + j]);
01289             }
01290         }
01291       for (j = 0; j < optimize->nvariables; ++j)
01292         {
01293           d = optimize->tolerance
01294             * (optimize->rangemax[j] - optimize->rangemin[j]);
01295           switch (optimize->algorithm)
01296             {
01297             case ALGORITHM_MONTE_CARLO:
01298               d *= 0.5;
01299               break;
01300             default:
01301               if (optimize->nsweeps[j] > 1)
01302                 d /= optimize->nsweeps[j] - 1;
01303               else
01304                 d = 0.;
01305             }
01306           optimize->rangemin[j] -= d;
01307           optimize->rangemin[j]
01308             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01309           optimize->rangemax[j] += d;
01310           optimize->rangemax[j]
01311             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01312           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01313                   optimize->rangemin[j], optimize->rangemax[j]);
01314           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01315                    optimize->label[j], optimize->rangemin[j],
01316                    optimize->rangemax[j]);
01317         }
01318 #if HAVE_MPI
01319       for (i = 1; i < ntasks; ++i)
01320         {
01321           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01322                     1, MPI_COMM_WORLD);
01323           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01324                     1, MPI_COMM_WORLD);
01325         }
01326     }
01327   else
01328     {
01329       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01330                 MPI_COMM_WORLD, &mpi_stat);
01331       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01332                 MPI_COMM_WORLD, &mpi_stat);
01333     }
01334 #endif
01335 #if DEBUG
01336   fprintf (stderr, "optimize_refine: end\n");
01337 #endif
```

```
01338 }
01339
01344 void
01345 optimize_step ()
01346 {
01347 #if DEBUG
01348   fprintf (stderr, "optimize_step: start\n");
01349 #endif
01350   optimize_algorithm ();
01351   if (optimize->nsteps)
01352     optimize_direction ();
01353 #if DEBUG
01354   fprintf (stderr, "optimize_step: end\n");
01355 #endif
01356 }
01357
01362 void
01363 optimize_iterate ()
01364 {
01365   unsigned int i;
01366 #if DEBUG
01367   fprintf (stderr, "optimize_iterate: start\n");
01368 #endif
01369   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01370   optimize->value_old = (double *)
01371     g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01372   optimize_step ();
01373   optimize_save_old ();
01374   optimize_refine ();
01375   optimize_print ();
01376   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01377     {
01378       optimize_step ();
01379       optimize_merge_old ();
01380       optimize_refine ();
01381       optimize_print ();
01382     }
01383 #if DEBUG
01384   fprintf (stderr, "optimize_iterate: end\n");
01385 #endif
01386 }
01387
01392 void
01393 optimize_free ()
01394 {
01395   unsigned int i, j;
01396 #if DEBUG
01397   fprintf (stderr, "optimize_free: start\n");
01398 #endif
01399   for (j = 0; j < optimize->ninputs; ++j)
01400     {
01401       for (i = 0; i < optimize->nexperiments; ++i)
01402         g_mapped_file_unref (optimize->file[j][i]);
01403       g_free (optimize->file[j]);
01404     }
01405   g_free (optimize->error_old);
01406   g_free (optimize->value_old);
01407   g_free (optimize->value);
01408   g_free (optimize->genetic_variable);
01409 #if DEBUG
01410   fprintf (stderr, "optimize_free: end\n");
01411 #endif
01412 }
01413
01418 void
01419 optimize_open ()
01420 {
01421   GTimeZone *tz;
01422   GDateTime *t0, *t;
01423   unsigned int i, j;
01424
01425 #if DEBUG
01426   char *buffer;
01427   fprintf (stderr, "optimize_open: start\n");
01428 #endif
01429
01430   // Getting initial time
01431 #if DEBUG
01432   fprintf (stderr, "optimize_open: getting initial time\n");
01433 #endif
01434   tz = g_time_zone_new_utc ();
01435   t0 = g_date_time_new_now (tz);
01436
01437   // Obtaining and initing the pseudo-random numbers generator seed
01438 #if DEBUG
01439   fprintf (stderr, "optimize_open: getting initial seed\n");
01440 #endif
```

```
01441   if (optimize->seed == DEFAULT_RANDOM_SEED)
01442     optimize->seed = input->seed;
01443   gsl_rng_set (optimize->rng, optimize->seed);
01444
01445   // Replacing the working directory
01446 #if DEBUG
01447   fprintf (stderr, "optimize_open: replacing the working directory\n");
01448 #endif
01449   g_chdir (input->directory);
01450
01451   // Getting results file names
01452   optimize->result = input->result;
01453   optimize->variables = input->variables;
01454
01455   // Obtaining the simulator file
01456   optimize->simulator = input->simulator;
01457
01458   // Obtaining the evaluator file
01459   optimize->evaluator = input->evaluator;
01460
01461   // Reading the algorithm
01462   optimize->algorithm = input->algorithm;
01463   switch (optimize->algorithm)
01464     {
01465     case ALGORITHM_MONTE_CARLO:
01466       optimize_algorithm = optimize_MonteCarlo;
01467       break;
01468     case ALGORITHM_SWEEP:
01469       optimize_algorithm = optimize_sweep;
01470       break;
01471     default:
01472       optimize_algorithm = optimize_genetic;
01473       optimize->mutation_ratio = input->mutation_ratio;
01474       optimize->reproduction_ratio = input->reproduction_ratio;
01475       optimize->adaptation_ratio = input->adaptation_ratio;
01476     }
01477   optimize->nvariables = input->nvariables;
01478   optimize->nsimulations = input->nsimulations;
01479   optimize->niterations = input->niterations;
01480   optimize->nbest = input->nbest;
01481   optimize->tolerance = input->tolerance;
01482   optimize->nsteps = input->nsteps;
01483   optimize->nestimates = 0;
01484   optimize->thresold = input->thresold;
01485   optimize->stop = 0;
01486   if (input->nsteps)
01487     {
01488       optimize->relaxation = input->relaxation;
01489       switch (input->direction)
01490         {
01491         case DIRECTION_METHOD_COORDINATES:
01492           optimize->nestimates = 2 * optimize->nvariables;
01493           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
01494           break;
01495         default:
01496           optimize->nestimates = input->nestimates;
01497           optimize_estimate_direction =
      optimize_estimate_direction_random;
01498         }
01499     }
01500
01501 #if DEBUG
01502   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01503 #endif
01504   optimize->simulation_best
01505     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01506   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01507
01508   // Reading the experimental data
01509 #if DEBUG
01510   buffer = g_get_current_dir ();
01511   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01512   g_free (buffer);
01513 #endif
01514   optimize->nexperiments = input->nexperiments;
01515   optimize->ninputs = input->experiment->ninputs;
01516   optimize->experiment
01517     = (char **) alloca (input->nexperiments * sizeof (char *));
01518   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01519   for (i = 0; i < input->experiment->ninputs; ++i)
01520     optimize->file[i] = (GMappedFile **)
01521       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01522   for (i = 0; i < input->nexperiments; ++i)
01523     {
01524 #if DEBUG
01525       fprintf (stderr, "optimize_open: i=%u\n", i);
```

```
01526 #endif
01527       optimize->experiment[i] = input->experiment[i].name;
01528       optimize->weight[i] = input->experiment[i].weight;
01529 #if DEBUG
01530       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01531                optimize->experiment[i], optimize->weight[i]);
01532 #endif
01533       for (j = 0; j < input->experiment->ninputs; ++j)
01534         {
01535 #if DEBUG
01536           fprintf (stderr, "optimize_open: template%u\n", j + 1);
01537 #endif
01538           optimize->file[j][i]
01539             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01540         }
01541     }
01542
01543   // Reading the variables data
01544 #if DEBUG
01545   fprintf (stderr, "optimize_open: reading variables\n");
01546 #endif
01547   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01548   j = input->nvariables * sizeof (double);
01549   optimize->rangemin = (double *) alloca (j);
01550   optimize->rangeminabs = (double *) alloca (j);
01551   optimize->rangemax = (double *) alloca (j);
01552   optimize->rangemaxabs = (double *) alloca (j);
01553   optimize->step = (double *) alloca (j);
01554   j = input->nvariables * sizeof (unsigned int);
01555   optimize->precision = (unsigned int *) alloca (j);
01556   optimize->nsweeps = (unsigned int *) alloca (j);
01557   optimize->nbits = (unsigned int *) alloca (j);
01558   for (i = 0; i < input->nvariables; ++i)
01559     {
01560       optimize->label[i] = input->variable[i].name;
01561       optimize->rangemin[i] = input->variable[i].rangemin;
01562       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01563       optimize->rangemax[i] = input->variable[i].rangemax;
01564       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01565       optimize->precision[i] = input->variable[i].precision;
01566       optimize->step[i] = input->variable[i].step;
01567       optimize->nsweeps[i] = input->variable[i].nsweeps;
01568       optimize->nbits[i] = input->variable[i].nbits;
01569     }
01570   if (input->algorithm == ALGORITHM_SWEEP)
01571     {
01572       optimize->nsimulations = 1;
01573       for (i = 0; i < input->nvariables; ++i)
01574         {
01575           if (input->algorithm == ALGORITHM_SWEEP)
01576             {
01577               optimize->nsimulations *= optimize->nsweeps[i];
01578 #if DEBUG
01579               fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01580                        optimize->nsweeps[i], optimize->nsimulations);
01581 #endif
01582             }
01583         }
01584     }
01585   if (optimize->nsteps)
01586     optimize->direction
01587       = (double *) alloca (optimize->nvariables * sizeof (double));
01588
01589   // Setting error norm
01590   switch (input->norm)
01591     {
01592     case ERROR_NORM_EUCLIDIAN:
01593       optimize_norm = optimize_norm_euclidian;
01594       break;
01595     case ERROR_NORM_MAXIMUM:
01596       optimize_norm = optimize_norm_maximum;
01597       break;
01598     case ERROR_NORM_P:
01599       optimize_norm = optimize_norm_p;
01600       optimize->p = input->p;
01601       break;
01602     default:
01603       optimize_norm = optimize_norm_taxicab;
01604     }
01605
01606   // Allocating values
01607 #if DEBUG
01608   fprintf (stderr, "optimize_open: allocating variables\n");
01609   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01610            optimize->nvariables, optimize->algorithm);
```

```
01611 #endif
01612   optimize->genetic_variable = NULL;
01613   if (optimize->algorithm == ALGORITHM_GENETIC)
01614     {
01615       optimize->genetic_variable = (GeneticVariable *)
01616         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01617       for (i = 0; i < optimize->nvariables; ++i)
01618         {
01619 #if DEBUG
01620           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01621                    i, optimize->rangemin[i], optimize->rangemax[i],
01622                    optimize->nbits[i]);
01623 #endif
01624           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
01625           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
01626           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01627         }
01628     }
01629 #if DEBUG
01630   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01631            optimize->nvariables, optimize->nsimulations);
01632 #endif
01633   optimize->value = (double *)
01634     g_malloc ((optimize->nsimulations
01635                + optimize->nestimates * optimize->nsteps)
01636               * optimize->nvariables * sizeof (double));
01637
01638   // Calculating simulations to perform for each task
01639 #if HAVE_MPI
01640 #if DEBUG
01641   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01642            optimize->mpi_rank, ntasks);
01643 #endif
01644   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
01645   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
01646   if (optimize->nsteps)
01647     {
01648       optimize->nstart_direction
01649         = optimize->mpi_rank * optimize->nestimates / ntasks;
01650       optimize->nend_direction
01651         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
01652     }
01653 #else
01654   optimize->nstart = 0;
01655   optimize->nend = optimize->nsimulations;
01656   if (optimize->nsteps)
01657     {
01658       optimize->nstart_direction = 0;
01659       optimize->nend_direction = optimize->nestimates;
01660     }
01661 #endif
01662 #if DEBUG
01663   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01664            optimize->nend);
01665 #endif
01666
01667   // Calculating simulations to perform for each thread
01668   optimize->thread
01669     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01670   for (i = 0; i <= nthreads; ++i)
01671     {
01672       optimize->thread[i] = optimize->nstart
01673         + i * (optimize->nend - optimize->nstart) / nthreads;
01674 #if DEBUG
01675       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01676                optimize->thread[i]);
01677 #endif
01678     }
01679   if (optimize->nsteps)
01680     optimize->thread_direction = (unsigned int *)
01681       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01682
01683   // Opening result files
01684   optimize->file_result = g_fopen (optimize->result, "w");
01685   optimize->file_variables = g_fopen (optimize->variables, "w");
01686
01687   // Performing the algorithm
01688   switch (optimize->algorithm)
01689     {
01690       // Genetic algorithm
01691     case ALGORITHM_GENETIC:
01692       optimize_genetic ();
```

```
01693       break;
01694
01695       // Iterative algorithm
01696     default:
01697       optimize_iterate ();
01698     }
01699
01700   // Getting calculation time
01701   t = g_date_time_new_now (tz);
01702   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01703   g_date_time_unref (t);
01704   g_date_time_unref (t0);
01705   g_time_zone_unref (tz);
01706   printf ("%s = %.6lg s\n",
01707           gettext ("Calculation time"), optimize->calculation_time);
01708   fprintf (optimize->file_result, "%s = %.6lg s\n",
01709            gettext ("Calculation time"), optimize->calculation_time);
01710
01711   // Closing result files
01712   fclose (optimize->file_variables);
01713   fclose (optimize->file_result);
01714
01715 #if DEBUG
01716   fprintf (stderr, "optimize_open: end\n");
01717 #endif
01718 }
```

5.19 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 Optimize

  Struct to define the optimization ation data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Functions

- 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.
- Optimize optimize [1]

  Optimization data.
- const xmlChar ∗ result_name

  Name of the result file.
- const xmlChar ∗ variables_name

  Name of the variables file.

5.19.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file optimize.h.

5.19.2 Function Documentation

5.19.2.1 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 461 of file optimize.c.

```
00462 {
00463   unsigned int i, j;
00464   double e;
00465 #if DEBUG
00466   fprintf (stderr, "optimize_best: start\n");
00467   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468            optimize->nsaveds, optimize->nbest);
```

```
00469 #endif
00470   if (optimize->nsaveds < optimize->nbest
00471       || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473       if (optimize->nsaveds < optimize->nbest)
00474         ++optimize->nsaveds;
00475       optimize->error_best[optimize->nsaveds - 1] = value;
00476       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477       for (i = optimize->nsaveds; --i;)
00478         {
00479           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00480             {
00481               j = optimize->simulation_best[i];
00482               e = optimize->error_best[i];
00483               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00484               optimize->error_best[i] = optimize->
      error_best[i - 1];
00485               optimize->simulation_best[i - 1] = j;
00486               optimize->error_best[i - 1] = e;
00487             }
00488           else
00489             break;
00490         }
00491     }
00492 #if DEBUG
00493   fprintf (stderr, "optimize_best: end\n");
00494 #endif
00495 }
```

5.19.2.2 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 786 of file optimize.c.

```
00787 {
00788 #if DEBUG
00789   fprintf (stderr, "optimize_best_direction: start\n");
00790   fprintf (stderr,
00791            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00792            simulation, value, optimize->error_best[0]);
00793 #endif
00794   if (value < optimize->error_best[0])
00795     {
00796       optimize->error_best[0] = value;
00797       optimize->simulation_best[0] = simulation;
00798 #if DEBUG
00799       fprintf (stderr,
00800                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00801                simulation, value);
00802 #endif
00803     }
00804 #if DEBUG
00805   fprintf (stderr, "optimize_best_direction: end\n");
00806 #endif
00807 }
```

5.19.2.3 void* optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 854 of file optimize.c.

```
00855 {
00856   unsigned int i, thread;
00857   double e;
00858 #if DEBUG
00859   fprintf (stderr, "optimize_direction_thread: start\n");
00860 #endif
00861   thread = data->thread;
00862 #if DEBUG
00863   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864            thread,
00865            optimize->thread_direction[thread],
00866            optimize->thread_direction[thread + 1]);
00867 #endif
00868   for (i = optimize->thread_direction[thread];
00869        i < optimize->thread_direction[thread + 1]; ++i)
00870     {
00871       e = optimize_norm (i);
00872       g_mutex_lock (mutex);
00873       optimize_best_direction (i, e);
00874       optimize_save_variables (i, e);
00875       if (e < optimize->thresold)
00876         optimize->stop = 1;
00877       g_mutex_unlock (mutex);
00878       if (optimize->stop)
00879         break;
00880 #if DEBUG
00881       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882 #endif
00883     }
00884 #if DEBUG
00885   fprintf (stderr, "optimize_direction_thread: end\n");
00886 #endif
00887   g_thread_exit (NULL);
00888   return NULL;
00889 }
```

Here is the call graph for this function:

5.19.2.4 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 928 of file optimize.c.

```
00930 {
00931   double x;
00932 #if DEBUG
00933   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934 #endif
00935   x = optimize->direction[variable];
00936   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938       if (estimate & 1)
00939         x += optimize->step[variable];
00940       else
00941         x -= optimize->step[variable];
00942     }
00943 #if DEBUG
00944   fprintf (stderr,
00945            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946            variable, x);
00947   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
```

```
00948 #endif
00949   return x;
00950 }
```

5.19.2.5 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 901 of file optimize.c.

```
00903 {
00904   double x;
00905 #if DEBUG
00906   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907 #endif
00908   x = optimize->direction[variable]
00909     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00910 #if DEBUG
00911   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00912            variable, x);
00913   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00914 #endif
00915   return x;
00916 }
```

5.19.2.6 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 1095 of file optimize.c.

```
01096 {
01097   unsigned int j;
01098   double objective;
01099   char buffer[64];
01100 #if DEBUG
01101   fprintf (stderr, "optimize_genetic_objective: start\n");
01102 #endif
01103   for (j = 0; j < optimize->nvariables; ++j)
01104     {
01105       optimize->value[entity->id * optimize->nvariables + j]
01106         = genetic_get_variable (entity, optimize->genetic_variable + j);
01107     }
01108   objective = optimize_norm (entity->id);
01109   g_mutex_lock (mutex);
01110   for (j = 0; j < optimize->nvariables; ++j)
01111     {
01112       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113       fprintf (optimize->file_variables, buffer,
01114                genetic_get_variable (entity, optimize->genetic_variable + j));
01115     }
01116   fprintf (optimize->file_variables, "%.14le\n", objective);
01117   g_mutex_unlock (mutex);
01118 #if DEBUG
01119   fprintf (stderr, "optimize_genetic_objective: end\n");
01120 #endif
01121   return objective;
01122 }
```

5.19.2.7 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 102 of file optimize.c.

```
00103 {
00104   unsigned int i;
00105   char buffer[32], value[32], *buffer2, *buffer3, *content;
00106   FILE *file;
00107   gsize length;
00108   GRegex *regex;
00109
00110 #if DEBUG
00111   fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114   // Checking the file
00115   if (!template)
00116     goto optimize_input_end;
00117
00118   // Opening template
00119   content = g_mapped_file_get_contents (template);
00120   length = g_mapped_file_get_length (template);
00121 #if DEBUG
00122   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124   file = g_fopen (input, "w");
00125
00126   // Parsing template
00127   for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG
00130       fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132       snprintf (buffer, 32, "@variable%u@", i + 1);
00133       regex = g_regex_new (buffer, 0, 0, NULL);
00134       if (i == 0)
00135         {
00136           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00137                                              optimize->label[i], 0, NULL);
00138 #if DEBUG
00139           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00140 #endif
00141         }
00142       else
00143         {
00144           length = strlen (buffer3);
00145           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00146                                              optimize->label[i], 0, NULL);
00147           g_free (buffer3);
00148         }
00149       g_regex_unref (regex);
00150       length = strlen (buffer2);
00151       snprintf (buffer, 32, "@value%u@", i + 1);
00152       regex = g_regex_new (buffer, 0, 0, NULL);
00153       snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->
      nvariables + i]);
00155
00156 #if DEBUG
00157       fprintf (stderr, "optimize_input: value=%s\n", value);
00158 #endif
00159       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                          0, NULL);
00161       g_free (buffer2);
00162       g_regex_unref (regex);
00163     }
00164
00165   // Saving input file
00166   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167   g_free (buffer3);
00168   fclose (file);
00169
00170 optimize_input_end:
00171 #if DEBUG
00172   fprintf (stderr, "optimize_input: end\n");
00173 #endif
00174   return;
00175 }
```

5.19.2.8 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 584 of file optimize.c.

```
00586 {
00587   unsigned int i, j, k, s[optimize->nbest];
00588   double e[optimize->nbest];
00589 #if DEBUG
00590   fprintf (stderr, "optimize_merge: start\n");
00591 #endif
00592   i = j = k = 0;
00593   do
00594     {
00595       if (i == optimize->nsaveds)
00596         {
00597           s[k] = simulation_best[j];
00598           e[k] = error_best[j];
00599           ++j;
00600           ++k;
00601           if (j == nsaveds)
00602             break;
00603         }
00604       else if (j == nsaveds)
00605         {
00606           s[k] = optimize->simulation_best[i];
00607           e[k] = optimize->error_best[i];
00608           ++i;
00609           ++k;
00610           if (i == optimize->nsaveds)
00611             break;
00612         }
00613       else if (optimize->error_best[i] > error_best[j])
00614         {
00615           s[k] = simulation_best[j];
00616           e[k] = error_best[j];
00617           ++j;
00618           ++k;
00619         }
00620       else
00621         {
00622           s[k] = optimize->simulation_best[i];
00623           e[k] = optimize->error_best[i];
00624           ++i;
00625           ++k;
00626         }
00627     }
00628   while (k < optimize->nbest);
00629   optimize->nsaveds = k;
00630   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631   memcpy (optimize->error_best, e, k * sizeof (double));
00632 #if DEBUG
00633   fprintf (stderr, "optimize_merge: end\n");
00634 #endif
00635 }
```

5.19.2.9 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 294 of file optimize.c.

```
00295 {
00296   double e, ei;
```

```
00297   unsigned int i;
00298 #if DEBUG
00299   fprintf (stderr, "optimize_norm_euclidian: start\n");
00300 #endif
00301   e = 0.;
00302   for (i = 0; i < optimize->nexperiments; ++i)
00303     {
00304       ei = optimize_parse (simulation, i);
00305       e += ei * ei;
00306     }
00307   e = sqrt (e);
00308 #if DEBUG
00309   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310   fprintf (stderr, "optimize_norm_euclidian: end\n");
00311 #endif
00312   return e;
00313 }
```

Here is the call graph for this function:

5.19.2.10 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 323 of file optimize.c.

```
00324 {
00325   double e, ei;
00326   unsigned int i;
00327 #if DEBUG
00328   fprintf (stderr, "optimize_norm_maximum: start\n");
00329 #endif
00330   e = 0.;
00331   for (i = 0; i < optimize->nexperiments; ++i)
00332     {
00333       ei = fabs (optimize_parse (simulation, i));
00334       e = fmax (e, ei);
00335     }
00336 #if DEBUG
00337   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338   fprintf (stderr, "optimize_norm_maximum: end\n");
00339 #endif
00340   return e;
00341 }
```

Here is the call graph for this function:

5.19.2.11 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 351 of file optimize.c.

```
00352 {
00353   double e, ei;
00354   unsigned int i;
00355 #if DEBUG
00356   fprintf (stderr, "optimize_norm_p: start\n");
00357 #endif
00358   e = 0.;
00359   for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361       ei = fabs (optimize_parse (simulation, i));
00362       e += pow (ei, optimize->p);
00363     }
00364   e = pow (e, 1. / optimize->p);
00365 #if DEBUG
00366   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00367   fprintf (stderr, "optimize_norm_p: end\n");
00368 #endif
00369   return e;
00370 }
```

Here is the call graph for this function:

5.19.2.12 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 380 of file optimize.c.

```
00381 {
00382   double e;
00383   unsigned int i;
00384 #if DEBUG
00385   fprintf (stderr, "optimize_norm_taxicab: start\n");
00386 #endif
00387   e = 0.;
00388   for (i = 0; i < optimize->nexperiments; ++i)
00389     e += fabs (optimize_parse (simulation, i));
00390 #if DEBUG
00391   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392   fprintf (stderr, "optimize_norm_taxicab: end\n");
00393 #endif
00394   return e;
00395 }
```

Here is the call graph for this function:

5.19.2.13 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 188 of file optimize.c.

```
00189 {
00190   unsigned int i;
00191   double e;
00192   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00193     *buffer3, *buffer4;
00194   FILE *file_result;
00195
00196 #if DEBUG
00197   fprintf (stderr, "optimize_parse: start\n");
00198   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00199            experiment);
00200 #endif
00201
00202   // Opening input files
00203   for (i = 0; i < optimize->ninputs; ++i)
00204     {
00205       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00206 #if DEBUG
00207       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00208 #endif
00209       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00210     }
00211   for (; i < MAX_NINPUTS; ++i)
00212     strcpy (&input[i][0], "");
00213 #if DEBUG
00214   fprintf (stderr, "optimize_parse: parsing end\n");
00215 #endif
00216
00217   // Performing the simulation
00218   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00219   buffer2 = g_path_get_dirname (optimize->simulator);
00220   buffer3 = g_path_get_basename (optimize->simulator);
00221   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00222   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00223             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00224             input[6], input[7], output);
00225   g_free (buffer4);
00226   g_free (buffer3);
00227   g_free (buffer2);
00228 #if DEBUG
00229   fprintf (stderr, "optimize_parse: %s\n", buffer);
00230 #endif
00231   system (buffer);
00232
00233   // Checking the objective value function
00234   if (optimize->evaluator)
00235     {
00236       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00237       buffer2 = g_path_get_dirname (optimize->evaluator);
00238       buffer3 = g_path_get_basename (optimize->evaluator);
00239       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00240       snprintf (buffer, 512, "\"%s\" %s %s %s",
00241                 buffer4, output, optimize->experiment[experiment], result);
00242       g_free (buffer4);
00243       g_free (buffer3);
00244       g_free (buffer2);
00245 #if DEBUG
00246       fprintf (stderr, "optimize_parse: %s\n", buffer);
00247 #endif
00248       system (buffer);
00249       file_result = g_fopen (result, "r");
00250       e = atof (fgets (buffer, 512, file_result));
00251       fclose (file_result);
00252     }
00253   else
00254     {
00255       strcpy (result, "");
00256       file_result = g_fopen (output, "r");
00257       e = atof (fgets (buffer, 512, file_result));
00258       fclose (file_result);
00259     }
00260
00261   // Removing files
00262 #if !DEBUG
00263   for (i = 0; i < optimize->ninputs; ++i)
00264     {
00265       if (optimize->file[i][0])
00266         {
00267           snprintf (buffer, 512, RM " %s", &input[i][0]);
00268           system (buffer);
00269         }
00270     }
00271   snprintf (buffer, 512, RM " %s %s", output, result);
00272   system (buffer);
00273 #endif
00274
```

```
00275   // Processing pending events
00276   show_pending ();
00277
00278 #if DEBUG
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
```

Here is the call graph for this function:

5.19.2.14 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 433 of file optimize.c.

```
00434 {
00435   unsigned int i;
00436   char buffer[64];
00437 #if DEBUG
00438   fprintf (stderr, "optimize_save_variables: start\n");
00439 #endif
00440   for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00443       fprintf (optimize->file_variables, buffer,
00444                optimize->value[simulation * optimize->
      nvariables + i]);
00445     }
00446   fprintf (optimize->file_variables, "%.14le\n", error);
00447 #if DEBUG
00448   fprintf (stderr, "optimize_save_variables: end\n");
00449 #endif
00450 }
```

5.19.2.15 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 959 of file optimize.c.

```
00960 {
00961   GThread *thread[nthreads_direction];
00962   ParallelData data[nthreads_direction];
00963   unsigned int i, j, k, b;
00964 #if DEBUG
00965   fprintf (stderr, "optimize_step_direction: start\n");
00966 #endif
00967   for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969       k = (simulation + i) * optimize->nvariables;
00970       b = optimize->simulation_best[0] * optimize->
      nvariables;
00971 #if DEBUG
00972       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00973                simulation + i, optimize->simulation_best[0]);
00974 #endif
00975       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00976         {
00977 #if DEBUG
00978           fprintf (stderr,
00979                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
```

```
00980                   i, j, optimize->value[b]);
00981 #endif
00982           optimize->value[k]
00983             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00984           optimize->value[k] = fmin (fmax (optimize->value[k],
00985                                            optimize->rangeminabs[j]),
00986                                      optimize->rangemaxabs[j]);
00987 #if DEBUG
00988           fprintf (stderr,
00989                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00990                    i, j, optimize->value[k]);
00991 #endif
00992         }
00993     }
00994   if (nthreads_direction == 1)
00995     optimize_direction_sequential (simulation);
00996   else
00997     {
00998       for (i = 0; i <= nthreads_direction; ++i)
00999         {
01000           optimize->thread_direction[i]
01001             = simulation + optimize->nstart_direction
01002             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01003             / nthreads_direction;
01004 #if DEBUG
01005           fprintf (stderr,
01006                    "optimize_step_direction: i=%u thread_direction=%u\n",
01007                    i, optimize->thread_direction[i]);
01008 #endif
01009         }
01010       for (i = 0; i < nthreads_direction; ++i)
01011         {
01012           data[i].thread = i;
01013           thread[i] = g_thread_new
01014             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01015         }
01016       for (i = 0; i < nthreads_direction; ++i)
01017         g_thread_join (thread[i]);
01018     }
01019 #if DEBUG
01020   fprintf (stderr, "optimize_step_direction: end\n");
01021 #endif
01022 }
```

Here is the call graph for this function:

5.19.2.16 void* optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 538 of file optimize.c.

```
00539 {
00540   unsigned int i, thread;
00541   double e;
00542 #if DEBUG
00543   fprintf (stderr, "optimize_thread: start\n");
00544 #endif
00545   thread = data->thread;
00546 #if DEBUG
00547   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00548            optimize->thread[thread], optimize->thread[thread + 1]);
00549 #endif
00550   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552       e = optimize_norm (i);
00553       g_mutex_lock (mutex);
00554       optimize_best (i, e);
```

```
00555       optimize_save_variables (i, e);
00556       if (e < optimize->thresold)
00557         optimize->stop = 1;
00558       g_mutex_unlock (mutex);
00559       if (optimize->stop)
00560         break;
00561 #if DEBUG
00562       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00563 #endif
00564     }
00565 #if DEBUG
00566   fprintf (stderr, "optimize_thread: end\n");
00567 #endif
00568   g_thread_exit (NULL);
00569   return NULL;
00570 }
```

Here is the call graph for this function:

5.20 optimize.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef OPTIMIZE__H
00039 #define OPTIMIZE__H 1
00040
00045 typedef struct
00046 {
00047   GMappedFile **file[MAX_NINPUTS];
00048   char **experiment;
00049   char **label;
00050   gsl_rng *rng;
00051   GeneticVariable *genetic_variable;
00053   FILE *file_result;
00054   FILE *file_variables;
00055   char *result;
00056   char *variables;
00057   char *simulator;
00058   char *evaluator;
00060   double *value;
00061   double *rangemin;
00062   double *rangemax;
00063   double *rangeminabs;
00064   double *rangemaxabs;
00065   double *error_best;
00066   double *weight;
00067   double *step;
00069   double *direction;
00070   double *value_old;
00072   double *error_old;
00074   unsigned int *precision;
00075   unsigned int *nsweeps;
00076   unsigned int *nbits;
```

```
00078   unsigned int *thread;
00080   unsigned int *thread_direction;
00083   unsigned int *simulation_best;
00084   double tolerance;
00085   double mutation_ratio;
00086   double reproduction_ratio;
00087   double adaptation_ratio;
00088   double relaxation;
00089   double calculation_time;
00090   double p;
00091   double thresold;
00092   unsigned long int seed;
00094   unsigned int nvariables;
00095   unsigned int nexperiments;
00096   unsigned int ninputs;
00097   unsigned int nsimulations;
00098   unsigned int nsteps;
00100   unsigned int nestimates;
00102   unsigned int algorithm;
00103   unsigned int nstart;
00104   unsigned int nend;
00105   unsigned int nstart_direction;
00107   unsigned int nend_direction;
00109   unsigned int niterations;
00110   unsigned int nbest;
00111   unsigned int nsaveds;
00112   unsigned int stop;
00113 #if HAVE_MPI
00114   int mpi_rank;
00115 #endif
00116 } Optimize;
00117
00122 typedef struct
00123 {
00124   unsigned int thread;
00125 } ParallelData;
00126
00127 // Global variables
00128 extern int ntasks;
00129 extern unsigned int nthreads;
00130 extern unsigned int nthreads_direction;
00131 extern GMutex mutex[1];
00132 extern void (*optimize_algorithm) ();
00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134                                               unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
00137 extern const xmlChar *result_name;
00138 extern const xmlChar *variables_name;
00139
00140 // Public functions
00141 void optimize_input (unsigned int simulation, char *input,
00142                      GMappedFile * template);
00143 double optimize_parse (unsigned int simulation, unsigned int experiment);
00144 double optimize_norm_euclidian (unsigned int simulation);
00145 double optimize_norm_maximum (unsigned int simulation);
00146 double optimize_norm_p (unsigned int simulation);
00147 double optimize_norm_taxicab (unsigned int simulation);
00148 void optimize_print ();
00149 void optimize_save_variables (unsigned int simulation, double error);
00150 void optimize_best (unsigned int simulation, double value);
00151 void optimize_sequential ();
00152 void *optimize_thread (ParallelData * data);
00153 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00154                      double *error_best);
00155 #if HAVE_MPI
00156 void optimize_synchronise ();
00157 #endif
00158 void optimize_sweep ();
00159 void optimize_MonteCarlo ();
00160 void optimize_best_direction (unsigned int simulation, double value);
00161 void optimize_direction_sequential ();
00162 void *optimize_direction_thread (ParallelData * data);
00163 double optimize_estimate_direction_random (unsigned int variable,
00164                                            unsigned int estimate);
00165 double optimize_estimate_direction_coordinates (unsigned int
      variable,
00166                                                 unsigned int estimate);
00167 void optimize_step_direction (unsigned int simulation);
00168 void optimize_direction ();
00169 double optimize_genetic_objective (Entity * entity);
00170 void optimize_genetic ();
00171 void optimize_save_old ();
00172 void optimize_merge_old ();
00173 void optimize_refine ();
00174 void optimize_step ();
00175 void optimize_iterate ();
```

```
00176 void optimize_free ();
00177 void optimize_open ();
00178
00179 #endif
```

5.21 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 <libintl.h>
#include <gtk/gtk.h>
#include "utils.h"
```

Include dependency graph for utils.c:

Functions

- void show_pending ()

  Function to show events on long computation.
- 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.
- char * error_message

  Error message.

5.21.1 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.

5.21.2 Function Documentation

5.21.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 329 of file utils.c.

```
00330 {
00331 #ifdef G_OS_WIN32
00332   SYSTEM_INFO sysinfo;
00333   GetSystemInfo (&sysinfo);
00334   return sysinfo.dwNumberOfProcessors;
00335 #else
00336   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00337 #endif
00338 }
```

5.21.2.2 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 353 of file utils.c.

```
00354 {
00355   unsigned int i;
00356   for (i = 0; i < n; ++i)
00357     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00358       break;
00359   return i;
00360 }
```

5.21.2.3 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 109 of file utils.c.

```
00110 {
00111   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00112 }
```

Here is the call graph for this function:

5.21.2.4 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 79 of file utils.c.

```
00080 {
00081 #if HAVE_GTK
00082   GtkMessageDialog *dlg;
00083
00084   // Creating the dialog
00085   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00086     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00087
00088   // Setting the dialog title
00089   gtk_window_set_title (GTK_WINDOW (dlg), title);
00090
00091   // Showing the dialog and waiting response
00092   gtk_dialog_run (GTK_DIALOG (dlg));
00093
00094   // Closing and freeing memory
00095   gtk_widget_destroy (GTK_WIDGET (dlg));
00096
00097 #else
00098   printf ("%s: %s\n", title, msg);
00099 #endif
00100 }
```

5.21.2.5 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 219 of file utils.c.

```
00220 {
00221   double x = 0.;
00222   xmlChar *buffer;
00223   buffer = xmlGetProp (node, prop);
00224   if (!buffer)
00225     *error_code = 1;
00226   else
00227     {
00228       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00229         *error_code = 2;
00230       else
00231         *error_code = 0;
00232       xmlFree (buffer);
00233     }
00234   return x;
00235 }
```

5.21.2.6 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 253 of file utils.c.

```
00255 {
00256   double x;
00257   if (xmlHasProp (node, prop))
00258     x = xml_node_get_float (node, prop, error_code);
00259   else
00260     {
00261       x = default_value;
00262       *error_code = 0;
00263     }
00264   return x;
00265 }
```

Here is the call graph for this function:

5.21.2.7 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 127 of file utils.c.

```
00128 {
00129   int i = 0;
00130   xmlChar *buffer;
00131   buffer = xmlGetProp (node, prop);
00132   if (!buffer)
00133     *error_code = 1;
00134   else
00135     {
00136       if (sscanf ((char *) buffer, "%d", &i) != 1)
00137         *error_code = 2;
00138       else
00139         *error_code = 0;
00140       xmlFree (buffer);
00141     }
00142   return i;
00143 }
```

5.21.2.8 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 158 of file utils.c.

```
00159 {
00160   unsigned int i = 0;
00161   xmlChar *buffer;
00162   buffer = xmlGetProp (node, prop);
00163   if (!buffer)
00164     *error_code = 1;
00165   else
00166     {
00167       if (sscanf ((char *) buffer, "%u", &i) != 1)
00168         *error_code = 2;
00169       else
00170         *error_code = 0;
00171       xmlFree (buffer);
00172     }
00173   return i;
00174 }
```

5.21.2.9 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 192 of file utils.c.

```
00194 {
00195   unsigned int i;
00196   if (xmlHasProp (node, prop))
00197     i = xml_node_get_uint (node, prop, error_code);
00198   else
00199     {
00200       i = default_value;
00201       *error_code = 0;
00202     }
00203   return i;
00204 }
```

Here is the call graph for this function:

5.21.2.10 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 316 of file utils.c.

```
00317 {
00318   xmlChar buffer[64];
00319   snprintf ((char *) buffer, 64, "%.14lg", value);
00320   xmlSetProp (node, prop, buffer);
00321 }
```

5.21.2.11 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 278 of file utils.c.

```
00279 {
00280   xmlChar buffer[64];
00281   snprintf ((char *) buffer, 64, "%d", value);
00282   xmlSetProp (node, prop, buffer);
00283 }
```

5.21.2.12 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 297 of file utils.c.

```
00298 {
00299   xmlChar buffer[64];
00300   snprintf ((char *) buffer, 64, "%u", value);
00301   xmlSetProp (node, prop, buffer);
00302 }
```

5.22 utils.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <unistd.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #if HAVE_GTK
00045 #include <gtk/gtk.h>
00046 #endif
00047 #include "utils.h"
00048
00049 #if HAVE_GTK
00050 GtkWindow *main_window;
00051 #endif
00052
00053 char *error_message;
00054
00059 void
00060 show_pending ()
00061 {
00062 #if HAVE_GTK
00063   while (gtk_events_pending ())
00064     gtk_main_iteration ();
00065 #endif
00066 }
00067
00078 void
00079 show_message (char *title, char *msg, int type)
00080 {
00081 #if HAVE_GTK
00082   GtkMessageDialog *dlg;
00083
00084   // Creating the dialog
```

```
00085   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00086     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00087
00088   // Setting the dialog title
00089   gtk_window_set_title (GTK_WINDOW (dlg), title);
00090
00091   // Showing the dialog and waiting response
00092   gtk_dialog_run (GTK_DIALOG (dlg));
00093
00094   // Closing and freeing memory
00095   gtk_widget_destroy (GTK_WIDGET (dlg));
00096
00097 #else
00098   printf ("%s: %s\n", title, msg);
00099 #endif
00100 }
00101
00108 void
00109 show_error (char *msg)
00110 {
00111   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00112 }
00113
00126 int
00127 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00128 {
00129   int i = 0;
00130   xmlChar *buffer;
00131   buffer = xmlGetProp (node, prop);
00132   if (!buffer)
00133     *error_code = 1;
00134   else
00135     {
00136       if (sscanf ((char *) buffer, "%d", &i) != 1)
00137         *error_code = 2;
00138       else
00139         *error_code = 0;
00140       xmlFree (buffer);
00141     }
00142   return i;
00143 }
00144
00157 unsigned int
00158 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00159 {
00160   unsigned int i = 0;
00161   xmlChar *buffer;
00162   buffer = xmlGetProp (node, prop);
00163   if (!buffer)
00164     *error_code = 1;
00165   else
00166     {
00167       if (sscanf ((char *) buffer, "%u", &i) != 1)
00168         *error_code = 2;
00169       else
00170         *error_code = 0;
00171       xmlFree (buffer);
00172     }
00173   return i;
00174 }
00175
00191 unsigned int
00192 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00193                                 unsigned int default_value, int *error_code)
00194 {
00195   unsigned int i;
00196   if (xmlHasProp (node, prop))
00197     i = xml_node_get_uint (node, prop, error_code);
00198   else
00199     {
00200       i = default_value;
00201       *error_code = 0;
00202     }
00203   return i;
00204 }
00205
00218 double
00219 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00220 {
00221   double x = 0.;
00222   xmlChar *buffer;
00223   buffer = xmlGetProp (node, prop);
00224   if (!buffer)
00225     *error_code = 1;
00226   else
00227     {
00228       if (sscanf ((char *) buffer, "%lf", &x) != 1)
```

```
00229         *error_code = 2;
00230       else
00231         *error_code = 0;
00232       xmlFree (buffer);
00233     }
00234   return x;
00235 }
00236
00252 double
00253 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00254                                  double default_value, int *error_code)
00255 {
00256   double x;
00257   if (xmlHasProp (node, prop))
00258     x = xml_node_get_float (node, prop, error_code);
00259   else
00260     {
00261       x = default_value;
00262       *error_code = 0;
00263     }
00264   return x;
00265 }
00266
00277 void
00278 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00279 {
00280   xmlChar buffer[64];
00281   snprintf ((char *) buffer, 64, "%d", value);
00282   xmlSetProp (node, prop, buffer);
00283 }
00284
00296 void
00297 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00298 {
00299   xmlChar buffer[64];
00300   snprintf ((char *) buffer, 64, "%u", value);
00301   xmlSetProp (node, prop, buffer);
00302 }
00303
00315 void
00316 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00317 {
00318   xmlChar buffer[64];
00319   snprintf ((char *) buffer, 64, "%.14lg", value);
00320   xmlSetProp (node, prop, buffer);
00321 }
00322
00328 int
00329 cores_number ()
00330 {
00331 #ifdef G_OS_WIN32
00332   SYSTEM_INFO sysinfo;
00333   GetSystemInfo (&sysinfo);
00334   return sysinfo.dwNumberOfProcessors;
00335 #else
00336   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00337 #endif
00338 }
00339
00340 #if HAVE_GTK
00341
00352 unsigned int
00353 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00354 {
00355   unsigned int i;
00356   for (i = 0; i < n; ++i)
00357     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00358       break;
00359   return i;
00360 }
00361
00362 #endif
```

5.23 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_pending ()

  Function to show events on long computation.
- 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.
- char * error_message

  Error message.

5.23.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file utils.h.

5.23.2 Function Documentation

5.23.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 329 of file utils.c.

```
00330 {
00331 #ifdef G_OS_WIN32
00332   SYSTEM_INFO sysinfo;
00333   GetSystemInfo (&sysinfo);
00334   return sysinfo.dwNumberOfProcessors;
00335 #else
00336   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00337 #endif
00338 }
```

5.23.2.2 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 353 of file utils.c.

```
00354 {
00355   unsigned int i;
00356   for (i = 0; i < n; ++i)
00357     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00358       break;
00359   return i;
00360 }
```

5.23.2.3 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 109 of file utils.c.

```
00110 {
00111   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00112 }
```

Here is the call graph for this function:

5.23.2.4 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 79 of file utils.c.

```
00080 {
00081 #if HAVE_GTK
00082   GtkMessageDialog *dlg;
00083
00084   // Creating the dialog
00085   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00086     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00087
00088   // Setting the dialog title
00089   gtk_window_set_title (GTK_WINDOW (dlg), title);
00090
00091   // Showing the dialog and waiting response
00092   gtk_dialog_run (GTK_DIALOG (dlg));
00093
00094   // Closing and freeing memory
00095   gtk_widget_destroy (GTK_WIDGET (dlg));
00096
00097 #else
00098   printf ("%s: %s\n", title, msg);
00099 #endif
00100 }
```

5.23.2.5 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 219 of file utils.c.

```
00220 {
00221   double x = 0.;
00222   xmlChar *buffer;
00223   buffer = xmlGetProp (node, prop);
00224   if (!buffer)
```

```
00225     *error_code = 1;
00226   else
00227     {
00228       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00229         *error_code = 2;
00230       else
00231         *error_code = 0;
00232       xmlFree (buffer);
00233     }
00234   return x;
00235 }
```

5.23.2.6 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 253 of file utils.c.

```
00255 {
00256   double x;
00257   if (xmlHasProp (node, prop))
00258     x = xml_node_get_float (node, prop, error_code);
00259   else
00260     {
00261       x = default_value;
00262       *error_code = 0;
00263     }
00264   return x;
00265 }
```

Here is the call graph for this function:

5.23.2.7 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 127 of file utils.c.

```
00128 {
00129   int i = 0;
00130   xmlChar *buffer;
00131   buffer = xmlGetProp (node, prop);
00132   if (!buffer)
```

```
00133     *error_code = 1;
00134   else
00135     {
00136       if (sscanf ((char *) buffer, "%d", &i) != 1)
00137         *error_code = 2;
00138       else
00139         *error_code = 0;
00140       xmlFree (buffer);
00141     }
00142   return i;
00143 }
```

5.23.2.8 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 158 of file utils.c.

```
00159 {
00160   unsigned int i = 0;
00161   xmlChar *buffer;
00162   buffer = xmlGetProp (node, prop);
00163   if (!buffer)
00164     *error_code = 1;
00165   else
00166     {
00167       if (sscanf ((char *) buffer, "%u", &i) != 1)
00168         *error_code = 2;
00169       else
00170         *error_code = 0;
00171       xmlFree (buffer);
00172     }
00173   return i;
00174 }
```

5.23.2.9 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 192 of file utils.c.

```
00194 {
00195   unsigned int i;
00196   if (xmlHasProp (node, prop))
```

```
00197     i = xml_node_get_uint (node, prop, error_code);
00198   else
00199     {
00200       i = default_value;
00201       *error_code = 0;
00202     }
00203   return i;
00204 }
```

Here is the call graph for this function:

5.23.2.10 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 316 of file utils.c.

```
00317 {
00318   xmlChar buffer[64];
00319   snprintf ((char *) buffer, 64, "%.14lg", value);
00320   xmlSetProp (node, prop, buffer);
00321 }
```

5.23.2.11 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 278 of file utils.c.

```
00279 {
00280   xmlChar buffer[64];
00281   snprintf ((char *) buffer, 64, "%d", value);
00282   xmlSetProp (node, prop, buffer);
00283 }
```

5.23.2.12 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 297 of file utils.c.

```
00298 {
00299   xmlChar buffer[64];
00300   snprintf ((char *) buffer, 64, "%u", value);
00301   xmlSetProp (node, prop, buffer);
00302 }
```

5.24 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef UTILS__H
00039 #define UTILS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 extern char *error_message;
00057
00058 // Public functions
00059 void show_pending ();
00060 void show_message (char *title, char *msg, int type);
00061 void show_error (char *msg);
00062 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00063 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00064                                 int *error_code);
00065 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00066                                              const xmlChar * prop,
00067                                              unsigned int default_value,
00068                                              int *error_code);
00069 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00070                            int *error_code);
00071 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
    ,
00072                                         double default_value, int *error_code);
00073 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00074 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00075                         unsigned int value);
00076 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00077 int cores_number ();
00078 #if HAVE_GTK
00079 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00080 #endif
00081
00082 #endif
```

5.25 variable.c File Reference

Source file to define the variable data.

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

Include dependency graph for variable.c:

Macros

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

  Macro to debug.

Functions

- void variable_new (Variable *variable)

  Function to create a new Variable struct.
- void variable_free (Variable *variable)

  Function to free the memory of a Variable struct.
- void variable_error (Variable *variable, char *message)

  Function to print a message error opening an Variable struct.
- int variable_open (Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.

Variables

- const char * format [NPRECISIONS]

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

  Array of variable precisions.

5.25.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file variable.c.

5.25.2 Function Documentation

5.25.2.1 void variable_error (Variable * *variable*, char * *message*)

Function to print a message error opening an Variable struct.

Parameters

variable	Variable struct.
message	Error message.

Definition at line 104 of file variable.c.

```
00105 {
00106   char buffer[64];
00107   if (!variable->name)
00108     snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109   else
00110     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111               message);
00112   error_message = g_strdup (buffer);
00113 }
```

5.25.2.2 void variable_free (Variable * *variable*)

Function to free the memory of a Variable struct.

Parameters

variable	Variable struct.

Definition at line 84 of file variable.c.

```
00085 {
00086 #if DEBUG
00087   fprintf (stderr, "variable_free: start\n");
00088 #endif
00089   xmlFree (variable->name);
00090 #if DEBUG
00091   fprintf (stderr, "variable_free: end\n");
00092 #endif
00093 }
```

5.25.2.3 void variable_new (Variable * *variable*)

Function to create a new Variable struct.

Parameters

variable	Variable struct.

Definition at line 66 of file variable.c.

```
00067 {
00068 #if DEBUG
00069   fprintf (stderr, "variable_new: start\n");
00070 #endif
00071   variable->name = NULL;
00072 #if DEBUG
00073   fprintf (stderr, "variable_new: end\n");
00074 #endif
00075 }
```

5.25.2.4 int variable_open (Variable * *variable*, xmlNode * *node*, unsigned int *algorithm*, unsigned int *nsteps*)

Function to open the variable file.

Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the direction search method.

Returns

1 on success, 0 on error.

Definition at line 130 of file variable.c.

```
00132 {
00133   int error_code;
00134
00135 #if DEBUG
00136   fprintf (stderr, "variable_open: start\n");
00137 #endif
00138
00139   variable->name = (char *) xmlGetProp (node, XML_NAME);
00140   if (!variable->name)
00141     {
00142       variable_error (variable, gettext ("no name"));
00143       goto exit_on_error;
00144     }
00145   if (xmlHasProp (node, XML_MINIMUM))
00146     {
00147       variable->rangemin = xml_node_get_float (node,
      XML_MINIMUM, &error_code);
00148       if (error_code)
00149         {
00150           variable_error (variable, gettext ("bad minimum"));
00151           goto exit_on_error;
00152         }
00153       variable->rangeminabs
00154         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MINIMUM,
00155                                            -G_MAXDOUBLE, &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, gettext ("bad absolute minimum"));
00159           goto exit_on_error;
00160         }
00161       if (variable->rangemin < variable->rangeminabs)
00162         {
00163           variable_error (variable, gettext ("minimum range not allowed"));
00164           goto exit_on_error;
00165         }
00166     }
00167   else
00168     {
00169       variable_error (variable, gettext ("no minimum range"));
00170       goto exit_on_error;
00171     }
00172   if (xmlHasProp (node, XML_MAXIMUM))
00173     {
00174       variable->rangemax = xml_node_get_float (node,
      XML_MAXIMUM, &error_code);
00175       if (error_code)
00176         {
00177           variable_error (variable, gettext ("bad maximum"));
00178           goto exit_on_error;
00179         }
00180       variable->rangemaxabs
00181         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MAXIMUM,
00182                                            G_MAXDOUBLE, &error_code);
00183       if (error_code)
00184         {
00185           variable_error (variable, gettext ("bad absolute maximum"));
00186           goto exit_on_error;
00187         }
00188       if (variable->rangemax > variable->rangemaxabs)
00189         {
00190           variable_error (variable, gettext ("maximum range not allowed"));
00191           goto exit_on_error;
00192         }
00193       if (variable->rangemax < variable->rangemin)
00194         {
00195           variable_error (variable, gettext ("bad range"));
```

```
00196           goto exit_on_error;
00197         }
00198     }
00199   else
00200     {
00201       variable_error (variable, gettext ("no maximum range"));
00202       goto exit_on_error;
00203     }
00204   variable->precision
00205     = xml_node_get_uint_with_default (node,
      XML_PRECISION,
00206                                       DEFAULT_PRECISION, &error_code);
00207   if (error_code || variable->precision >= NPRECISIONS)
00208     {
00209       variable_error (variable, gettext ("bad precision"));
00210       goto exit_on_error;
00211     }
00212   if (algorithm == ALGORITHM_SWEEP)
00213     {
00214       if (xmlHasProp (node, XML_NSWEEPS))
00215         {
00216           variable->nsweeps
00217             = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00218           if (error_code || !variable->nsweeps)
00219             {
00220               variable_error (variable, gettext ("bad sweeps"));
00221               goto exit_on_error;
00222             }
00223         }
00224       else
00225         {
00226           variable_error (variable, gettext ("no sweeps number"));
00227           goto exit_on_error;
00228         }
00229 #if DEBUG
00230       fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00231 #endif
00232     }
00233   if (algorithm == ALGORITHM_GENETIC)
00234     {
00235       // Obtaining bits representing each variable
00236       if (xmlHasProp (node, XML_NBITS))
00237         {
00238           variable->nbits = xml_node_get_uint (node,
      XML_NBITS, &error_code);
00239           if (error_code || !variable->nbits)
00240             {
00241               variable_error (variable, gettext ("invalid bits number"));
00242               goto exit_on_error;
00243             }
00244         }
00245       else
00246         {
00247           variable_error (variable, gettext ("no bits number"));
00248           goto exit_on_error;
00249         }
00250     }
00251   else if (nsteps)
00252     {
00253       variable->step = xml_node_get_float (node, XML_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
00255         {
00256           variable_error (variable, gettext ("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG
00262   fprintf (stderr, "variable_open: end\n");
00263 #endif
00264   return 1;
00265
00266 exit_on_error:
00267   variable_free (variable);
00268 #if DEBUG
00269   fprintf (stderr, "variable_open: end\n");
00270 #endif
00271   return 0;
00272 }
```

Here is the call graph for this function:

5.25.3 Variable Documentation

5.25.3.1 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 49 of file variable.c.

5.25.3.2 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 54 of file variable.c.

5.26 variable.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include "utils.h"
00045 #include "variable.h"
00046
00047 #define DEBUG 0
00048
00049 const char *format[NPRECISIONS] = {
00050   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00051   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00052 };
```

```
00053
00054 const double precision[NPRECISIONS] = {
00055   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,
00056   1e-13, 1e-14
00057 };
00058
00065 void
00066 variable_new (Variable * variable)
00067 {
00068 #if DEBUG
00069   fprintf (stderr, "variable_new: start\n");
00070 #endif
00071   variable->name = NULL;
00072 #if DEBUG
00073   fprintf (stderr, "variable_new: end\n");
00074 #endif
00075 }
00076
00083 void
00084 variable_free (Variable * variable)
00085 {
00086 #if DEBUG
00087   fprintf (stderr, "variable_free: start\n");
00088 #endif
00089   xmlFree (variable->name);
00090 #if DEBUG
00091   fprintf (stderr, "variable_free: end\n");
00092 #endif
00093 }
00094
00103 void
00104 variable_error (Variable * variable, char *message)
00105 {
00106   char buffer[64];
00107   if (!variable->name)
00108     snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109   else
00110     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111               message);
00112   error_message = g_strdup (buffer);
00113 }
00114
00129 int
00130 variable_open (Variable * variable, xmlNode * node, unsigned int algorithm,
00131                unsigned int nsteps)
00132 {
00133   int error_code;
00134
00135 #if DEBUG
00136   fprintf (stderr, "variable_open: start\n");
00137 #endif
00138
00139   variable->name = (char *) xmlGetProp (node, XML_NAME);
00140   if (!variable->name)
00141     {
00142       variable_error (variable, gettext ("no name"));
00143       goto exit_on_error;
00144     }
00145   if (xmlHasProp (node, XML_MINIMUM))
00146     {
00147       variable->rangemin = xml_node_get_float (node,
      XML_MINIMUM, &error_code);
00148       if (error_code)
00149         {
00150           variable_error (variable, gettext ("bad minimum"));
00151           goto exit_on_error;
00152         }
00153       variable->rangeminabs
00154         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MINIMUM,
00155                                            -G_MAXDOUBLE, &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, gettext ("bad absolute minimum"));
00159           goto exit_on_error;
00160         }
00161       if (variable->rangemin < variable->rangeminabs)
00162         {
00163           variable_error (variable, gettext ("minimum range not allowed"));
00164           goto exit_on_error;
00165         }
00166     }
00167   else
00168     {
00169       variable_error (variable, gettext ("no minimum range"));
00170       goto exit_on_error;
00171     }
```

```
00172   if (xmlHasProp (node, XML_MAXIMUM))
00173     {
00174       variable->rangemax = xml_node_get_float (node,
      XML_MAXIMUM, &error_code);
00175       if (error_code)
00176         {
00177           variable_error (variable, gettext ("bad maximum"));
00178           goto exit_on_error;
00179         }
00180       variable->rangemaxabs
00181         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MAXIMUM,
00182                                            G_MAXDOUBLE, &error_code);
00183       if (error_code)
00184         {
00185           variable_error (variable, gettext ("bad absolute maximum"));
00186           goto exit_on_error;
00187         }
00188       if (variable->rangemax > variable->rangemaxabs)
00189         {
00190           variable_error (variable, gettext ("maximum range not allowed"));
00191           goto exit_on_error;
00192         }
00193       if (variable->rangemax < variable->rangemin)
00194         {
00195           variable_error (variable, gettext ("bad range"));
00196           goto exit_on_error;
00197         }
00198     }
00199   else
00200     {
00201       variable_error (variable, gettext ("no maximum range"));
00202       goto exit_on_error;
00203     }
00204   variable->precision
00205     = xml_node_get_uint_with_default (node,
      XML_PRECISION,
00206                                       DEFAULT_PRECISION, &error_code);
00207   if (error_code || variable->precision >= NPRECISIONS)
00208     {
00209       variable_error (variable, gettext ("bad precision"));
00210       goto exit_on_error;
00211     }
00212   if (algorithm == ALGORITHM_SWEEP)
00213     {
00214       if (xmlHasProp (node, XML_NSWEEPS))
00215         {
00216           variable->nsweeps
00217             = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00218           if (error_code || !variable->nsweeps)
00219             {
00220               variable_error (variable, gettext ("bad sweeps"));
00221               goto exit_on_error;
00222             }
00223         }
00224       else
00225         {
00226           variable_error (variable, gettext ("no sweeps number"));
00227           goto exit_on_error;
00228         }
00229 #if DEBUG
00230       fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00231 #endif
00232     }
00233   if (algorithm == ALGORITHM_GENETIC)
00234     {
00235       // Obtaining bits representing each variable
00236       if (xmlHasProp (node, XML_NBITS))
00237         {
00238           variable->nbits = xml_node_get_uint (node,
      XML_NBITS, &error_code);
00239           if (error_code || !variable->nbits)
00240             {
00241               variable_error (variable, gettext ("invalid bits number"));
00242               goto exit_on_error;
00243             }
00244         }
00245       else
00246         {
00247           variable_error (variable, gettext ("no bits number"));
00248           goto exit_on_error;
00249         }
00250     }
00251   else if (nsteps)
00252     {
00253       variable->step = xml_node_get_float (node, XML_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
```

```
00255         {
00256           variable_error (variable, gettext ("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG
00262   fprintf (stderr, "variable_open: end\n");
00263 #endif
00264   return 1;
00265
00266 exit_on_error:
00267   variable_free (variable);
00268 #if DEBUG
00269   fprintf (stderr, "variable_open: end\n");
00270 #endif
00271   return 0;
00272 }
```

5.27 variable.h File Reference

Header file to define the variable data.

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

Data Structures

- struct Variable

  Struct to define the variable data.

Enumerations

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

  Enum to define the algorithms.

Functions

- void variable_new (Variable *variable)

  Function to create a new Variable struct.
- void variable_free (Variable *variable)

  Function to free the memory of a Variable struct.
- void variable_error (Variable *variable, char *message)

  Function to print a message error opening an Variable struct.
- int variable_open (Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.

Variables

- const char * format [NPRECISIONS]

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

  Array of variable precisions.

5.27.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Definition in file variable.h.

5.27.2 Enumeration Type Documentation

5.27.2.1 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 variable.h.

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

5.27.3 Function Documentation

5.27.3.1 void variable_error (Variable * *variable*, char * *message*)

Function to print a message error opening an Variable struct.

Parameters

variable	Variable struct.
message	Error message.

Definition at line 104 of file variable.c.

```
00105 {
00106   char buffer[64];
00107   if (!variable->name)
00108     snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109   else
00110     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111               message);
00112   error_message = g_strdup (buffer);
00113 }
```

5.27.3.2 void variable_free (Variable * *variable*)

Function to free the memory of a Variable struct.

Parameters

variable	Variable struct.

Definition at line 84 of file variable.c.

```
00085 {
00086 #if DEBUG
00087   fprintf (stderr, "variable_free: start\n");
00088 #endif
00089   xmlFree (variable->name);
00090 #if DEBUG
00091   fprintf (stderr, "variable_free: end\n");
00092 #endif
00093 }
```

5.27.3.3 void variable_new (Variable * *variable*)

Function to create a new Variable struct.

Parameters

variable	Variable struct.

Definition at line 66 of file variable.c.

```
00067 {
00068 #if DEBUG
00069   fprintf (stderr, "variable_new: start\n");
00070 #endif
00071   variable->name = NULL;
00072 #if DEBUG
00073   fprintf (stderr, "variable_new: end\n");
00074 #endif
00075 }
```

5.27.3.4 int variable_open (Variable * *variable*, xmlNode * *node*, unsigned int *algorithm*, unsigned int *nsteps*)

Function to open the variable file.

Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the direction search method.

Returns

1 on success, 0 on error.

Definition at line 130 of file variable.c.

```
00132 {
00133   int error_code;
00134
00135 #if DEBUG
00136   fprintf (stderr, "variable_open: start\n");
00137 #endif
00138
00139   variable->name = (char *) xmlGetProp (node, XML_NAME);
00140   if (!variable->name)
00141     {
00142       variable_error (variable, gettext ("no name"));
00143       goto exit_on_error;
00144     }
00145   if (xmlHasProp (node, XML_MINIMUM))
00146     {
```

```
00147       variable->rangemin = xml_node_get_float (node,
      XML_MINIMUM, &error_code);
00148       if (error_code)
00149         {
00150           variable_error (variable, gettext ("bad minimum"));
00151           goto exit_on_error;
00152         }
00153       variable->rangeminabs
00154         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MINIMUM,
00155                                            -G_MAXDOUBLE, &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, gettext ("bad absolute minimum"));
00159           goto exit_on_error;
00160         }
00161       if (variable->rangemin < variable->rangeminabs)
00162         {
00163           variable_error (variable, gettext ("minimum range not allowed"));
00164           goto exit_on_error;
00165         }
00166     }
00167   else
00168     {
00169       variable_error (variable, gettext ("no minimum range"));
00170       goto exit_on_error;
00171     }
00172   if (xmlHasProp (node, XML_MAXIMUM))
00173     {
00174       variable->rangemax = xml_node_get_float (node,
      XML_MAXIMUM, &error_code);
00175       if (error_code)
00176         {
00177           variable_error (variable, gettext ("bad maximum"));
00178           goto exit_on_error;
00179         }
00180       variable->rangemaxabs
00181         = xml_node_get_float_with_default (node,
      XML_ABSOLUTE_MAXIMUM,
00182                                            G_MAXDOUBLE, &error_code);
00183       if (error_code)
00184         {
00185           variable_error (variable, gettext ("bad absolute maximum"));
00186           goto exit_on_error;
00187         }
00188       if (variable->rangemax > variable->rangemaxabs)
00189         {
00190           variable_error (variable, gettext ("maximum range not allowed"));
00191           goto exit_on_error;
00192         }
00193       if (variable->rangemax < variable->rangemin)
00194         {
00195           variable_error (variable, gettext ("bad range"));
00196           goto exit_on_error;
00197         }
00198     }
00199   else
00200     {
00201       variable_error (variable, gettext ("no maximum range"));
00202       goto exit_on_error;
00203     }
00204   variable->precision
00205     = xml_node_get_uint_with_default (node,
      XML_PRECISION,
00206                                       DEFAULT_PRECISION, &error_code);
00207   if (error_code || variable->precision >= NPRECISIONS)
00208     {
00209       variable_error (variable, gettext ("bad precision"));
00210       goto exit_on_error;
00211     }
00212   if (algorithm == ALGORITHM_SWEEP)
00213     {
00214       if (xmlHasProp (node, XML_NSWEEPS))
00215         {
00216           variable->nsweeps
00217             = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00218           if (error_code || !variable->nsweeps)
00219             {
00220               variable_error (variable, gettext ("bad sweeps"));
00221               goto exit_on_error;
00222             }
00223         }
00224       else
00225         {
00226           variable_error (variable, gettext ("no sweeps number"));
00227           goto exit_on_error;
00228         }
```

```
00229 #if DEBUG
00230       fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00231 #endif
00232     }
00233   if (algorithm == ALGORITHM_GENETIC)
00234     {
00235       // Obtaining bits representing each variable
00236       if (xmlHasProp (node, XML_NBITS))
00237         {
00238           variable->nbits = xml_node_get_uint (node,
      XML_NBITS, &error_code);
00239           if (error_code || !variable->nbits)
00240             {
00241               variable_error (variable, gettext ("invalid bits number"));
00242               goto exit_on_error;
00243             }
00244         }
00245       else
00246         {
00247           variable_error (variable, gettext ("no bits number"));
00248           goto exit_on_error;
00249         }
00250     }
00251   else if (nsteps)
00252     {
00253       variable->step = xml_node_get_float (node, XML_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
00255         {
00256           variable_error (variable, gettext ("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG
00262   fprintf (stderr, "variable_open: end\n");
00263 #endif
00264   return 1;
00265
00266 exit_on_error:
00267   variable_free (variable);
00268 #if DEBUG
00269   fprintf (stderr, "variable_open: end\n");
00270 #endif
00271   return 0;
00272 }
```

Here is the call graph for this function:

5.28 variable.h

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

```
00039 #define VARIABLE__H 1
00040
00045 enum Algorithm
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
00051
00056 typedef struct
00057 {
00058   char *name;
00059   double rangemin;
00060   double rangemax;
00061   double rangeminabs;
00062   double rangemaxabs;
00063   double step;
00064   unsigned int precision;
00065   unsigned int nsweeps;
00066   unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open (Variable * variable, xmlNode * node, unsigned int algorithm,
00077                    unsigned int nsteps);
00078
00079 #endif
```

Index