EDACC
User Guide
version 0.1

Abstract

We present the main capabilities of EDACC and describe how to use EDACC for managing solvers and instances, create experiments with them, launch them on different computer clusters, monitor them and then analyze the results.

Contents

1 Outline

Here we will have an overview of this user guide specifying where the user can find what!

2 Introduction

2.1 General Terms

To keep this user-guide consistent we would like to define a couple of terms that will be often used through this document. Even if you are familiar with these, we recommend you to take a short look at them.

Algorithm
Example 1:
We define an **algorithm** as an arbitrary computation method.Examples of well known algorithms are the family of sorting algorithms like bubble-sort, quick-sort or merge-sort.

Solver
The concrete implementation of an algorithm in an arbitrary programming language is called a **solver**, which normally has an input and an output.

Instance
Example 2:
A solver is designed to solve a certain type of problem.One concrete problem (an instantiation of it) is called a (problem) **instance** . For the sorting algorithms an example of an instance would be a file containing a sequence of number that has to be sorted.

Solver Parameters
To control the behavior of a solver it can have parameters which we will call **solver parameters**. These parameters can also be seen as an input of the solver which is normally passed through the command line. For example the quick-sort algorithm could have a parameter "pivot" that can take the values $\{left, right, random\}$. With the help of this parameter the behavior of the solver can be controlled regarding how it should choose the pivot element during sorting.

Solver Configuration
A solver together with a fixed set of values for its parameters is called a **solver configuration**. Randomized quick-sort would be a solver configuration of the quick-sort solver with the parameter "pivot" set to *random*.

Computing System
To see how a solver performs on a certain instance we need to execute that solver. For this task we need a **computing system** which in **EDACC** ca be a single computer, computer cluster of even a grid.

Instance Property
Result Property
As **EDACC** provides a wide variety of statistical analysis tools we need a way to point out different forms of informations. We define an **instance property** as any kind of information that can be extracted from an instance. The output of a solver is called the **result** and any information that can be computed from the result is called **result property**.

2.2 Motivation

Algorithm engineering:
When designing and implementing algorithms one is at the end of the process confronted with the problem of evaluating the implementation on the targeted problem set. As the authors of **EDACC** are familiar with algorithms for the satisfiability problem (SAT) we will take this sort of algorithms as further examples. After designing and implementing a SAT solver we would like to see how it performs on a set of instances (let us suppose that our solver is an implementation of a stochastic one i. e.,the result of the solver on the same instance will be a random variable).

Normally we would start our solver on each instance and record the runtime or some quality measure. This is a sequential process and could be easily performed with the help of simple shell script. But there are some questions that have to be answered before starting the evaluation process.

1. How long is the solver allowed to compute on one instance? And how do we restrict that?
2. In the case of randomized solvers, how often do we call the solver on each problem set?
3. Do we limit the resources used by the solver (i. e.,maximum of memory, maximum stack size)?

Example 3:

Let us now suppose we would like to test our SAT-solver on 100 instances where we allow a timeout of 200 seconds. Because of the stochastic nature of the solver we are going to run it for 100 times on each instances. We are not going to limit other resources. Now we get a set of (100 instances) $\times$ (100 runs) that produces a set of 10000 jobs. Having a timeout limit of 200 seconds our computation could take up to $10000 \cdot 200 = 2000000 sec \cong 24 days$ on a single CPU machine in worst case.

Now everybody has access to multi-core machines or even some clusters with multiple CPU's. So we could speed up the computation by using this sort of resources but then we get the problem of equally spreading our jobs. And more than that we have to collect the results after that and process them with some statistical tools.

Most of the researchers solve this problems by writing a collection of scripts. This solution is error-prone and time consuming because there is no very simple way to equally spread jobs across multiple machines. Collecting the results and merging them together can also yield a not negligible amount of work. One more disadvantage is that the results can be seldom reproduced without having the complete set of scripts and even then there might be some steps that are not incorporated within the scripts.

EDACC features

To solve this problems we have designed **EDACC**. The main goal of **EDACC** are to:

1. manage solvers and instances and archiving them in a database with the help of a GUI
2. create experiment settings by configuring solvers and selecting the instances
3. evaluating the jobs of an experiment on arbitrary many machines
4. provide analysis tools for the results
5. provide an online tool to monitor and analyze experiments

2.3 EDACC Components

The four major components of **EDACC** are the:

1. Grapical user interface (GUI)
2. Database (DB)
3. Compute client (client)
4. Web frontend (WF) (optional)

2.4 System Requirements

! → 1. GUI - Sun Java 6 (JRE 6), optional: R (see Experiment Mode - README.txt for more details)

! → 2. Database - MySQL version 5.1 or above, tested with version 5.1.41 on Ubuntu. The machine the database runs on is the most important factor of the performance of **EDACC**. The following components will have the greatest impact on database performance:

   - The more RAM MySQL can use, the less it has to access slow hard disks on read-transactions. It also enables MySQL to keep indexes and whole tables in memory. This will greatly affect the ability to work on multiple experiments at the same time.
   - Hard disk performance is not as important as RAM but all data has to be written to the disk eventually which is when fast access time and write throughput become important.
   - A fast multi-core CPU will enable MySQL to handle more requests concurrently but is not as important as RAM.

   Network latency and bandwidth should also be considered when the GUI and clients are run on remote machines. The clients will write the output of solvers and metadata back to the database so the required bandwidth depends on the size of the generated output and metadata.

! → 3. Client - see section 5.2

! → 4. Web Frontend - see section 6.2

2.5 Getting started

To use **EDACC** you will have to follow these steps:

1. Set up a mysql database (see 2.5.1.
2. Download the latest **EDACC** GUI from sourceforge.org (eventually check for updates within **EDACC**).

2.5.1 MySQL Installation and Setup

MySQL configuration

MySQL installation is simple on most Linux distributions. On Ubuntu, for example, you have to type `apt-get install mysql-server` and set a root account password when the installation procedure asks you to. After installation there are a few settings that have to be adjusted in order to use MySQL with **EDACC**. These can be found in the configuration file my.cnf usually located at `/etc/mysql/my.cnf`. Adjust the following settings:

```
[mysqld] # look for this section
# listen on all IPs/allow network connections :
bind-address = 0.0.0.0
 # maximum packet size (important for large instances):
max_allowed_packet = 2048M

# comment out the skip-networking directive,
# if present:
#skip-networking
```

```
# increase session timeout
# and maximum number of simultaneous connections
wait_timeout = 259200
max_connections = 1000

# performance related settings
# innodb_buffer_pool_size is the most important parameter
# set this to as much RAM as you can spare on the machine:
innodb_buffer_pool_size = 1024M
```

Creating databases

After saving the modifications, restart your MySQL server (Ubuntu: `service mysql restart`) and open a MySQL client session by typing `mysql -uroot -p` which will then ask you for the root password you specified during MySQL installation. In the MySQL client shell you can then create an empty database that can be used as **EDACC** database by running the following commands:

```
CREATE DATABASE edacc;
GRANT ALL PRIVILEGES ON edacc.* TO 'edaccuser'@'%'
  IDENTIFIED BY 'dbuserpassword' WITH GRANT OPTION;
```

This will create an empty database called *edacc* and grant the MySQL user *edaccuser* with the password *dbuserpassword* all necessary rights. In the **EDACC** GUI, client and Web Frontend you can then use this account when connecting to the database.

2.5.2 Starting the GUI

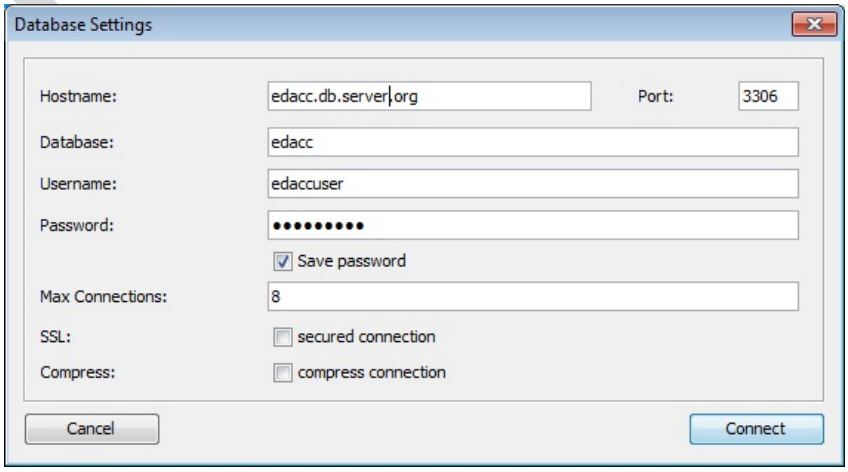
If you have succeeded to set up a database now you can start the GUI of **EDACC** by typing:

```
java -jar EDACC.jar
```

3 Graphical User Interface

3.1 Database connection

Every time you will start **EDACC** you will be prompted to provide the connection data to the MySQL™ database you would like to work with.



Host name / IP

In the connection dialog you have to provide the host name or the IP-address of your MySQL™ database.

Port — If you have configured the MySQL™ server to use an other port then the default MySQL™ port 3306, you can specify this in the **Port:** text field.

DB name — Further you also have to provide a valid database name and a user along with the corresponding password.

Save password — If you would like to save the password of this connections for further usage you can check the **Save password** check-box. **EDACC** will save the password for you in a configuration file. The password is saved in plain text, so if other users have access to your private files they will be able to read the password from the configurtion file.

! →

Max Connections — **EDACC** is a multi-threaded program and will use more than on connection to the database to speed up certain tasks. We recommend to allow up to 8 simultaneous database connections, but if you have restrictions on this number you can specify it in the **Max Connection:** text field. **TODO:**Simon: wenn das weniger sind?

SSL connections — If you are going to use **EDACC** to store trusted data we strongly recommend to enable a SSL connection by checking the **secured connection** check box. **TODO:**Be aware that this kind of connection is only possible is the MySQL™ server is configured accordingly.

Compressed Connections — When working with **EDACC** through a slow network connection you might want to turn on compression by checking the **compress connection** check box.

Connect — After providing all the information you can connect to the MySQL™ server.

Create DB — When you connect the first time to a database **EDACC** will create for you all the needed tables. **TODO:**Was passiert wenn man sich an einer falschen DB verbindet, die ein anderes Modell hat. Kann man das EDACC DB Modell neu erzeugen über einen bestehenden?

DB Model version — As **EDACC** is under full development and the database model may be extended to support new features, **EDACC** will check if the database model is compatible with the GUI version. Within this check we differentiate between two cases:

DB Model upgrade

1. The database model version is to old for the GUI. In this case **EDACC** will offer you the possibility to upgrade your database scheme to the latest version. **TODO:**Werden da auch mehrere versionen übersprungen? Sprich von version 0.1 auf 0.4?

GUI update

2. The database model version is to new for the GUI. In this case you should update the GUI. You can do this by using the automatic update function of **EDACC**, which can be found under **Help** → **Check for Updates**. Another possibility is to download the latest release form the project site at `http://sourceforge.net/projects/edacc/`.

3.2 Modes

EDACC is split up in two modes:

1. Manage DB Mode
2. Experiment Mode

There is a strict split-up between these two modes. You can be only in one mode when working with **EDACC**. When starting **EDACC** you will always be in the the manage DB mode, which will allow you to manage

your solvers and instances before creating experiments with them. To switch between modes you have to choose the desired mode from the menu bar **Mode**.

3.3 Manage DB Mode

The manage DB Mode is again split up in two parts: solvers and instances.

3.3.1 Solvers

Solver As mentioned in section 2.1 a solver is a program which implements an algorithm. In **EDACC** we store the following information about a solver:

Solver name
1. A human-readable name of the solver.

Solver version
2. The version number of the solver. The combination of name and version must be unique.

Solver description
3. A short description of the solver.

Solver authors
4. The list of the authors of a solver.

Solver code
5. The sourcecode of the solver.

Solver binaries
6. A solver can consist of different binaries, which have the same source code but differ in the compile options (eg. the architecture) or the chosen compiler version. There must be at least one solver binary.

Parameters Every solver has a list of several parameters which control its behaviour. To build a valid parameter list string, EDACC needs the following information:

name The human-readable internal name of the parameter. This name has no effect to the generated command-line and is only needed for reasons of indentification in the EDACC system.

prefix The parameter prefix defines how the parameter is called on the command-line. The Unix program `ls` for example has a parameter with the prefix `-l`.

Boolean Some parameters don't have an actual value but act as switches for a certain functionality of a solver. The `-l` parameter of the Unix program `ls` for example is such a boolean parameter.

Mandatory Some parameters need to be specified to start the solver binary. Such parameters are called mandatory.

Space

Order Some solvers need a special order of the parameters. This order is specified by an ascending number. The parameter with the smallest number will be used first in the command-line string. If two parameters have the same order number, the order between those two parameters doesn't matter.

Add Solver By clicking the button "New" in the solver panel a new empty line in the solver table is created. To fill the new entry with information fill in the form below the table with the static information of the solver. Optionally you can attach the code of the solver to the entry by clicking on "Add Code" and choosing the files or directories from your file system.

! → To create a valid solver entry, it is necessary to specify at least one solver binary.

Add Solver Binary The table below the text fields with the static solver information shows the solver binaries which are already attached to the chosen solver. To add another binary, click on the "Add" button below the table with the binaries. Choose the binary files which are needed to run the solver from your file system. EDACC then tries to zip the chosen files. This can take a few seconds.

To complete the process, some information on the binary have to be given:

Alternative Binary Name A human-readable name of the binary. This information is only needed that the binary can be recognized by the user in the program.

Execution File The main file of the binary, which will be called by the EDACC client to start the binary. You can choose it from the list of the previously chosen binary files. For default, the first file is chosen.

Additional run command Some binaries or scripts need a special command to start them (this is very usual for interpreted languages or scripts). For example a Java JAR archive can be started by the additional run command `java -jar`. A preview of the command executed on the grid by the client is shown in the text line below the text field for the addtional run command.

Version The version string specifies for example the architecture of the compiled binary or the used compiler. The version of the underlying source code is specified in the solver information, which is described above!

Click on "Add binary" to complete the process.

! → All modifications on solvers, solver binaries or parameters are not directly saved to the database. To persist your changes, you can choose the button "Save To DB".

Edit Solver To edit the information of a certain solver, choose the solver from the solver table. The text fields below the table will show the currently saved information of the solver. By changing those values, the information in the solver table will be adjusted automatically.

Edit Solver Binary

Delete Solver Binary
! → To delete a solver binary, choose it in the list of binaries and click on the "Delete"-Button below the table. After confirming the delete action, the solver binary will be removed directly from the database!

Delete Solver If you want to delete a solver with all attached information, code, binaries and parameter, click on the "Delete"-Button in the solver panel. The solver will be removed directly from the database, after confirming the delete action. To delete multiple solvers at once, just hold `Ctrl` in the solver table.

Add Parameter To add a parameter to a solver, choose the solver from the solver table. On the parameters panel, the list of parameters will show all parameters of the chosen solver. By clicking on "New" in the parameter panel, a new empty line will appear in the parameters table and is selected automatically. The text fields and checkboxes below the tab show the default values created for the new parameter. To change them, simply change

the values in those control fields. The information in the table will adjust automatically. For your comfort, the order value will be incremented automatically by creating a new parameter. Changes on the parameter panel won't take effect until you chose the button "Save To DB".

! →

Edit Parameter

If you want to edit the information of a parameter, first chose the solver whose parameters you want to edit from the solver table. Then coose the parameter you like to edit and modify the information in the text fields below the table. Click "Save To DB" to persist your changes in the database.

Delete Parameter

To delete parameters of a solver, choose the solver and the parameter you want to delete (by holding `Ctrl` in the parameter table, you can select multiple parameters). Click on "Delete" in the parameter panel. The delete action is performed immediately on the database! All your changes will be lost!

! →

Save changes to DB

Adding and Editing solver, binary or parameter information will take effect to the database by choosing the button "Save To DB".

Export

To export the solver code and the binaries of a solver, choose the solver from the solver table and click on "Export". After selecting the directory where the exported files should be saved, EDACC creates a bunch of zip files with the exported code and binaries from the database.

Reload from DB

If you like to undo your changes you haven't already commited to the database by choosing "Save To DB", you can click on "Reload from DB". This has the effect that all information in the program will be stashed and reloaded from the database, so your uncommited changes will be lost.

3.3.2 Instances

Instance

An instance is a practical instantiation of a problem. The instances tab provides functions for the user to add, remove, generate and organize instances.

Instance Class

Instance classes enables the user to group and organize instances into different categories. It's possible that an instance is assigned to several instance classes. An instance classes can include other instance classes and are represented as an tree.

Add Instance

To add one or more instances, use the button Add. In the following dialogue are four possible choices.

1. If automatic class generation is selected, the added instances are added to instance classes which are generated from the dependent on the directory of the instances to add. .
2. If the automatic class generation isn't selected, the user have to choose an instance class from the instance class table of the dialogue. Else if automatic class generation is selected, the choice of an instance class is optional.
3. To save the instances as compressed files in the database, select Compress.
4. In the field File Extension, the user has to define the file extension of the instance files to add.

To continue the process, the user has to use the button Ok and select the directory of the instances or their explicit files. This depends on the decisions made in the previous dialogue.

Remove Instance — Use the button Remove under the instance table, to remove instances from the database.

Generate Instance — ?

Export Instance — The export function of instances from EDACC is provided by the Button Export, located on the left side under the instance table. The user has to choose the directory, in which the instances are exported.

Compute Property — To compute a property of a group of instances, the user has to select the instances in the table and use the button Compute Property. After pressing the button a new dialogue is shown, with the possible properties to compute. To start the computation process, a property has to be chosen and the button Compute to be pressed.

Filter Instances — The user can call the filter function dialogue of the instance table by using the button Filter. ....

Select Columns of Instances — A selection of columns of the instance table can be called by pressing the button Select Columns. The appearing dialogue shows two kinds of columns which can be selected by the user, the Basic Columns and the Instance Property Columns. The variety of property columns depends on the number of defined instance properties.

Add Instance to Instance Class — The user has to select a group of instances, use the button Add to Class and select the instance class to which the instances have to be added to in the appeared dialogue.

Remove Instance from Instance Class — To remove a group of instances from a instance class, the user has to select these instances and press the button Remove from Class.

3.4 Experiment Mode

3.4.1 Experiments

Experiment — An experiment consists of solver configurations, instances and the number of runs for each solver configuration and instance. In the experiment tab the user can create/remove/edit experiments.

Create — By using the create-button in the first tab of the experiment mode an experiment can be created. This will open a dialog where you have to provide some data.

1. Name: the name for the new experiment
2. Description: a description for the experiment. Provide some useful information about the experiment to quickly identify experiments in the experiments table.
3. Default Cost: this will be the default cost for this experiment. This will affect some default behaviour in the GUI and the WF, e.g. the appropriate column in the job browser will be visible by default and the others will not be visible. The user can choose between three types of costs:
   (a) resultTime: the CPU time needed for a run will be used as cost.
   (b) wallTime: the real time needed for a run will be used as cost.

(c) cost: if a verifier is used which outputs cost, then this will be used as cost.

4. Limits: the user can specify if the outputs should be limited. Outputs that can be limited are solver output, watcher output and verifier output. This might save disk space on the DB server. It is possible to preserve the first and/or the last lines or bytes.

5. Configuration experiment: if set, this will be a configuration experiment and the Configuration Scenario tab will be enabled for this experiment, see section 3.4.3 for more information.

After pressing the create-button the newly created experiment will be loaded automatically.

Remove To remove an experiment use the appropriate button.

Edit To edit an experiment use the appropriate button. There you can edit the data you provided by creating the experiment. If you want to change the priority of an experiment you can do this by directly editing this property in the experiment table. The same applies to activating and deactivating experiments. For more details about the effect of the priority property, see section **??**. Deactivated experiments won't be computed by clients.

Discard To discard an experiment use the appropriate button. This button is only available if an experiment is loaded.

Load To load an experiment use the appropriate button or double click the experiment you want to load in the experiment table.

Import It is possible to import data from other experiments. To import data from other experiments the following steps have to be applied:

1. Load the experiment you want to import data to
2. Press the import button in the experiment tab. This will open a new window with three tables for experiments, solver configurations and instances.
3. Select the experiments you want to import data from. This will update the solver configuration and instance tables to show all solver configurations and instances for the selected experiments. Orange rows mean that the solver configuration or instance in that row exists in the currently loaded experiment. Two solver configurations are considered as equal if they have the same solver binary associated and have the same launch parameters.
4. Select the solver configurations and instances to import
5. Select *import finished jobs* if you also want to import jobs
6. Press *Import*

! → Note that this action might generate new jobs. This *might* happen if you import solver configurations and instances with their jobs to an experiment where some of the solver configurations and instances actually exist and they are in the *same seed group*.

Filter

3.4.2 Client Browser

The client browser represents all clients currently connected to the database.

Dead clients Red rows denote dead clients. A client is considered to be dead if the

client didn't communicate with the database for a period of time.

The client browser also deals as the only way to directly communicate with clients.

Kill clients

After selecting the clients you can open the context menu with the right mouse button and select *Kill Clients Hard* or *Kill Clients Soft*. Hard means that the clients will terminate all currently computing jobs and sign off. Soft means that the clients won't start new jobs and will wait for the currently computing jobs to finish.

Client details

To view the jobs which a client has computed in his lifetime you can double click a client entry in the client table. This will show a dialog with a table containing all jobs the client calculated and is currently calculating. You can also send messages to the clients in this dialog.

3.4.3 Configuration Scenario (Optional)

Configuration Scenarios are used to define a solver with its parameters to be configured with a configurator. This tab is only enabled for configuration experiments, see section 3.4.1. To define a configuration scenario the DB must contain at least one solver with a parameter graph specified (see section **??**). There are two ways to specify a configuration scenario. Either the user imports the configuration scenario from another configuration scenario in the DB by using the **Import Scenario** button or the user specifies the configuration scenario manually. The following steps have to be applied to specify a configuration scenario manually:

Parameter Graph

Import Scenario

1. Select the solver to be used for this configuration scenario in the **Solver** combo box. This combo box will only contain solvers which have parameter graphs specified. If it contains no solvers then you might want to head to the DB mode and specify a parameter graph for one or more solvers.
2. Select the solver binary for the selected solver that will be used to execute the solver on the grid in the **Solver Binary** combo box.
3. Select the parameters that should be used for the configurator. First the user has to select the parameters that should be used for the solver configurations created by the configurator. Then the user must specify which of these parameters should be configured and which have fixed values. If a parameter should have a fixed value and is not a boolean parameter then the value should also be specified by the user.

After the steps above the configuration scenario can be saved. It is now possible to generate some random solver configurations to test the configuration scenario. To do this the user has to use the **Generate Solver Configurations** button. An input dialog will pop up and the user can choose the number of solver configurations that should be created. After applying the dialog the solver configurations will be created and saved to the DB. The user can see the result in the **Solvers** tab.

Generate Solver Configurations

Instance, Seed-Course

The last task which can be done in this tab is generating an Instance, Seed-Course for the configuration experiment. After selecting the instances for the solver to be trained on in the **Instances** tab (see section 3.4.5) the user can generate an Instance, Seed-Course by using the appropriate button. This will open a dialog with a table containing the course. The user can move/sort the instances to change that course. By selecting the columns with some instance properties by using the

Columns button the user has the ability to sort the instances after an instance property.

3.4.4 Solvers

Choosing solvers

Creating solver configurations is done in the solvers tab. This tab contains a table on the left side and a panel with all solver configurations currently associated with this experiment. To create solver configurations you have to choose solvers for which you want create solver configurations. This can be done in the left table, the solvers table. By selecting some solvers and finally pressing the *choose*-button, solver configuration prototypes will be created for the solvers. You can see the newly created solver configurations in the panel on the right side. This panel is organized as follows. For each solver exists one layer. Each layer contains all solver configuration for the associated solver. A solver configuration is titled with a name. The name can be changed and is used in the other areas of the GUI to identify a solver configuration. So it might be good practice to choose different names for the solver configurations in an experiment.

Modifying solver configurations

A solver configuration consists of a solver binary, parameters and a seed group. The solver binary is chosen in the first combo box. The parameters can be specified in the parameters table. Just select the parameters you want for this solver configuration and specify their values if they have some. Finally you have to specify the seed group. The default seed group is *0*. You might want to change that. See section **??** for more information about seed groups.

Importing solver configurations

To import solver configurations from other experiments you can import them in the experiments tab (see section 3.4.1) or if you just want to import solver configurations without their jobs, you can use the *Import*-button. This will open a dialog where you can import solver configurations where you have two options. Either you want to import solver configurations from one or more experiments or you want to import solver configurations from one or more solvers. Choose the tab in this dialog accordingly. After selecting the solver configurations to be imported, use the *Import*-button of this dialog to import the selected solver configurations.

! → The imported solver configurations will not be saved to the DB until the user uses the *Save*-button.

Tabular view for solver configurations

To change the view of the solver configuration panel to a tabular view, press the *Change View*-button. This will change the panel into a table. Here you can remove multiple solver configurations by selecting them and opening the context menu by pressing the right mouse button and choosing *Remove*. It is also possible to edit solver configurations in that view by double clicking a solver configuration or by using the context menu. If you select a solver configuration in the tabular view and change back to the normal view then the view will automatically be scrolled to the previously selected solver configuration.

! → All modifications to solver configurations are not directly saved to the DB. You can always use the *Undo*-button the undo all changes and load the last saved state. By pressing the *Save*-button all modified and new solver configurations will be saved to the DB and deleted solver configurations will be removed from the DB.

! → Modifying and saving solver configurations which have calculated runs might be not a good idea. Therefore the GUI supplies a possibility to reset

the affected jobs. This might not be needed if the changed parameters have no effects to the results.

3.4.5 Instances

Instances are associated with an experiment in the *Instances* tab. This tab consists of two tables. On the left side are the instance classes and on the right side are the instances which are in the selected instance classes.

To associate instances to the currently loaded experiment, you can use the buttons below the instances table or select the instances manually.

Filter

Additionally it is possible to filter instances by using the *Filter*-button and sort the instances by clicking on the appropriate column in the table header. With the *Columns*-button other columns can be made visible, e.g. columns for instance properties. The *Undo*-button can be used at any time to revert changes to the last saved state. To make changes permanent the *Save*-button must be used.

Columns

Undo

Save

Import

It is also possible to import the selection from other experiments by using the import functionality in the experiments tab, see section 3.4.1.

3.4.6 Generate Jobs

After choosing solver configurations and selecting instances for the experiment, jobs can be generated in the *Generate Jobs* tab. In this tab there is a table representing a matrix with the instances and solver configurations in the experiment. Each cell in the table represents the number of jobs for the instances in that row and the solver configuration in that column.

To set the number of jobs for all or the selected cells, you can use the *Set Number of Runs*-button. This will open a dialog where you can choose if you want to set the number of runs for all or only the selected cells. With the *Number of Runs*-text field you can choose the number of runs and finish this process by using the *Apply*-button. Now you can determine which cells have be changed. White cells means no change, red cells means that the value is below the actually saved value and green cells means that the value is above the actually saved value. By using the *Generate Jobs*-button, those changes can be made permanent, and jobs will be created and/or deleted accordingly. This will open a dialog where you have to provide some data:

Colors

Generate Jobs

1. Timeout: the CPU time limit for the newly created jobs in seconds.
2. Max memory: the maximum amount of memory the newly created jobs can use in megabytes.
3. Wall clock time limit: the maximum real time the newly created jobs can use in seconds.
4. Stack size limit: the maximum stack size the newly created jobs can use in megabytes.
5. Max Seed: if seeds have to be generated (i.e. there are solver configurations which have a seed parameter) then this will be the maximum seed.

If the value *-1* is given for the limits, it means that there is no limit. After using the *Generate*-button the jobs will be generated. The results of this process can be revised in the *Job Browser* tab, see section 3.4.8.

Queue Selection
: By using the *Select Queue*-button it is possible to select one or more grid queues for computation. This will open a dialog where you can select grid queues for this experiment. For more information about creating grid queues see section 3.4.7.

Generate Cluster Package
: ..

3.4.7 Grid Queues

3.4.8 Job Browser

3.4.9 Analysis

4 Parameter search space specification

This section explains the concept of parameter graphs that are used to encode the parameter space of solvers in EDACC. If you are only interested in specifying the parameter space of a solver we suggest to skim over the details and first take a look at the example 4.1. In the context of parameter spaces a parameter is an input variable of a program and is defined by a name and a domain. Properties of parameters such as the command line prefix and the order in which the should appear when calling the solver executable aren't of interest in this context.

Definition 4.1:
A domain defines the set of possible values that can be assigned to a parameter (in a solver configuration). It can be one of the following or the union (which we call mixed domain) of any number of them (except for the flag domain, which can only occur on its own).

1. real: values between a lower and an upper bound
2. integer: values between a lower and an upper bound
3. ordinal: list of values in a min to max order
4. categorial: set of possible values
5. optional: consists only of a special value "not specified"
6. flag: consists of two special values "on" and "off" (for parameters that are flags, i.e. present or not)

Definition 4.2:
The parameter space of a solver is defined by its parameters and their possible values. The parameter space can be further constrained by dependencies between parameters such as

1. Parameter X can be specified if parameter Y takes on certain values
2. Parameter X has to be specified if parameter Y takes on certain values
3. Parameter X has to take on certain values depending on the values of parameters Y, Z, ...

There are several problems that come up in the context of EDACC: Determine if a given solver configuration is valid, i.e. in the solver's parameter space. Given the parameter space, construct a valid solver configuration. Given a valid solver configuration, find a "neighbouring" solver configuration that is also valid.

Definition 4.3:
A parameter graph is a directed, acyclic graph that represents the parameter space. It consists of AND-Nodes and OR-Nodes and edges between them. Edges are directed and allowed only between different types of nodes. OR-Nodes can have multiple incoming edges, while AND-Nodes can only have exactly one incoming edge. Additionally edges have a group number which is 0 if the edge doesn't belong to any group. Parameter graphs have a single unique AND-Node without any incoming edges. This node will be referred to as start node.

Definition 4.4:
OR-Nodes have a reference to a parameter.

Definition 4.5:
AND-Nodes have a domain and a parameter reference to the same parameter as the preceding OR-node. AND-Nodes partition the possible values of the parameter that they (and the preceding OR-node) reference. The domain of an AND-Node has to be a subset of the domain of the preceding OR-Node.

The general idea is that the parameter space is specified by following the structure of the graph from the start node and constraining the parameters using the domains encountered on the nodes. AND-Nodes imply that all outgoing edges have to be followed while OR-Nodes mean that exactly one edge has to be followed.

More formally:

Definition 4.6:
A solver configuration is valid if the start node (an AND-Node) of the parameter graph is satisfied. Satisfied means:

1. an AND-Node is satisfied if the corresponding parameter value lies in its domain and all OR-nodes adjacent via ungrouped edges are satisfied.
2. an OR-Node is satisfied if exactly one adjacent AND-Node is satisfied and for at least one set of incoming edges with common group number the preceding AND-Nodes are all satisfied.

4.1 Example

Example 4:

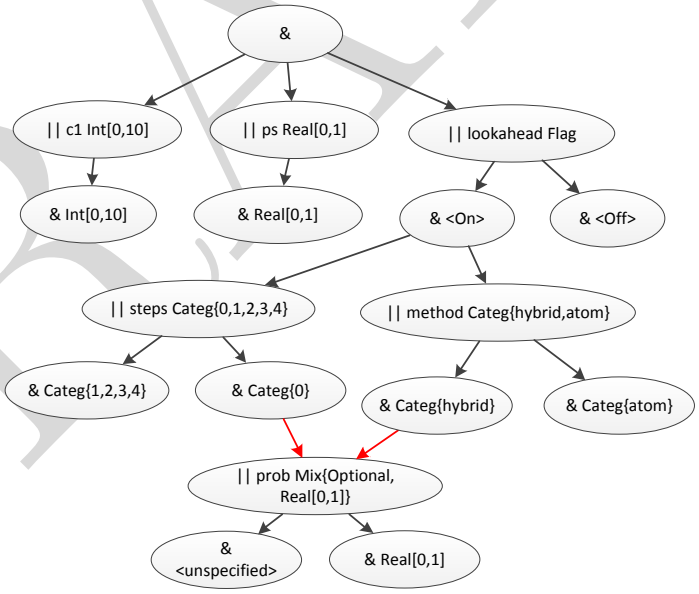
Consider a solver that has the following parameters:

- *c1* which takes on integer values in $[1, 10]$.
- *ps* which takes on real values in $[0, 1]$.
- A flag called *lookahead* which can be present or not.
- A categorical parameter *steps* which takes on values in $\{0, 1, 2, 3, 4\}$.
- Another categorical parameter *method* whose value is either "hybrid" or "atom".
- A parameter *prob* which can be left out or take on real values in $[0, 1]$.

Furthermore there are some restrictions and requirements:

- Both *c1* and *ps* have to be always specified.
- If the *lookahead* flag is present, both *steps* and *method* have to be present.
- If *steps* takes on the value 0 and *method* takes on the value "hybrid", then the parameter *prob* can take on values in its real domain $[0, 1]$ or be left out.

This parameter space can be encoded in a parameter graph as defined earlier in the following way:



The two red edges imply the membership of the edges to the same edge group $\neq 0$. Black edges mean that the edge doesn't belong to any group. For simplicity, the parameter references of AND-Nodes (to the same parameter as the preceeding OR-Node) are not shown in the graph.

5 Client

5.1 Introduction

The computation is client is used to compute the jobs of experiments. Usually there have to be a lot of jobs computed to evaluate experiments and since they are independent from each other, this task can be parallelized across many CPU cores. The computation client can be started on arbitrarily many machines and will manage the available CPUs and start jobs from the available experiments. It connects to the central database and downloads all required resources such as instances and solver binaries and writes back the results to the database.

5.2 System requirements

The client is written in C/C++ and should be able to run on most Linux distributions where a MySQL C connector library is available.

Because the central storage location for all required computational ressources, experiment metadata and results is a MySQL database, the client has to be able to establish a connection to the machine that hosts the database.

TCP/IP Connections:

This means that the machines where the client runs on have to be able to establish a TCP/IP connection to the database machine.

The client was mainly tested on the bwGRID[1], a distributed computer cluster that consists of several hundred nodes at several physical locations at universities of Baden-Württemberg, Germany. Even though the machine hardware is homogenous, the network topology of bwGRID is not. In cases where direct network access from the computation nodes back to the database server is not possible it is usually possible to tunnel a connection over the cluster's login node back to the database via SSH.

Connection alternatives:

For example running

Example 5:

```
ssh -f -N -L 0.0.0.0:1234:databasehost:3306 user@databasehost
```

on the login node sets up a tunnel for connections at port 1234 and forwards them to your database machine at port 3306, where MySQL listens. The options `-f` and `-N` will let the tunnel continue to run in the background, even after logging out from the login node. To terminate the tunnel, simply run e.g. `killall ssh` on the login node. In the client configuration (see below), you would then specify the IP/hostname of the login node and port 1234 as the database hostname and port.

Other than that, the client has to be able to write temporary files to some location on the filesystem. This can be configured (5.3.1) if it differs from the client binary location.

Because the client will download missing solver binaries and instances and upload results it also needs a reasonably fast network connection to the database. Shared filesystems can considerably reduce the required bandwidth since every file is only downloaded once. Alternatively you can create a package from within the GUI that contains all solver binaries and instances of an experiment. However, if you modify experiments while the client is running it will still download missing files.

Shared filesystems:

[1] http://www.bw-grid.de/

5.3 Usage

5.3.1 Configuration

Configuration file:

Configuration is done by some command line arguments and a simple configuration file, called "config". This file has to be in the **working directory** of the client at runtime. In the configuration file you have to specify the database connection details and which hardware the client runs on. This is done by configuring so called "grid queues" in the GUI application. They contain some basic information about the computation hardware such as number of CPUs per machine. The client will then use this information to run as many parallel jobs as the grid queue information allows it on each machine where it is launched. Here is a sample configuration file:

Example 6:

```
host = database.host.foo.com
port = 3306
username = dbusername
password = dbpassword
database = dbname
gridqueue = 3
verifier = ./verifiers/SAT
```

Note that the gridqueue value is simply the ID of the grid queue. Another (optional) configuration option is the verifier line. It tells the client if it should run a program on the output that a solver generated. For example, if your experiments consist of attempting to solve boolean propositional logic formulas you can use a SAT verifier that tests if the solution given by a solver is actually correct. The verifier configuration option is simply a path to an executable that the client calls with standardized parameters. See section 5.4 for more information on verifiers.

Command line arguments:

Beside the configuration file there are several command line options the client accepts, please also see "./client –help":

```
-v <verbosity>:
  Integer value between 0 and 4 (from lowest to
  highest verbosity)
-l:
  If flag is set, the log output is written to a file
  instead of stdout.
-w <wait for jobs time (s)>:
  How long the client should wait for jobs after
  it didn't get any new jobs before exiting.
-i <handle workers interval ms>:
  How long the client should wait after handling
  workers and before looking for a new job.
-k:
  Whether to keep the solver and watcher output files after
  uploading to the DB. Default behaviour is to delete them.
-b <path>:
  Base path for creating temporary directories and files.
-h:
  Toggles whether the client should continue to run even
  though the CPU hardware of the grid queue is not homogenous.
-s:
  Enables simulation mode where the client will fetch and run
  jobs but won't write any results back to the database.
```

Verbosity controls the amount of log output the client generates. A value of 4 is only useful for debugging purposes, a value of 0 will make the client log important messages and all errors.

If the "l" flag is set, log output goes to a file whose name includes the hostname and IP address of the machine the client runs on. This is done to avoid name clashes in shared filesystems typically found in computer clusters. Otherwise log output goes to standard output.

With the "-w" option you can tell the client how long to wait before exiting after it didn't start any jobs. This can be useful to keep the clients running and ready to process new jobs while you evaluate preliminary results and add new jobs or whole experiments. The wait option is also used to determine how long attempts should be made to reconnect to the database after connection losses. The default value is 10 seconds.

The "-i" option controls how long the client should wait between its main processing loop iterations. If this value is low, it will look for new jobs when there are unused CPUs more frequently. For maximum job throughput this value should be lower than the average job processing time but lower values will also put more strain on the database and increase the client's CPU usage. The default value is 100ms which should work fine in most cases. The client will also adapt to situations where there are free CPUs but no more jobs and increase the interval internally and fall back to the configured value once it got another job.

The "-k" flag tells the client that it should keep temporary job output files after a job is finished. The default behaviour is to delete them.

The "-b" base path option can be used to specify a directory the client can use to write temporary files to. The default value is ".", i.e. the working directory at runtime.

inhomogenous machines

! → The first client to start with a particular grid queue will write the information about the machine it runs on to the grid queue entry in the database. All following clients will then compare their machines to the information in the grid queue and exit, unless the number of cores and the CPU model name match. With the "-h" option you can override this behaviour.

5.3.2 Launching

After configuration you can simply run the client on your computation machines. On computer clusters there are often queuing systems that you have to use to gain access to the nodes. On bwGRID for example, we could use the following short PBS (portable batch script) and submit (*qsub scriptname*) our client to a node with 8 cores:

```
#!/bin/sh
#PBS -l walltime=10:00:00
#PBS -l nodes=1:ppn=8
cd /path/to/shared/fs/with/client/executable
./client -v0 -l -i200 -w120
```

! → You should always run the client from within its directory (i.e. cd to the directory) to avoid problems with relative paths such as the verifier path from the example configuration above.

As soon as clients start you should be able to see jobs changing their status from "not started" to "running" in the GUI's or Web frontend's

job browsers.

5.3.3 Troubleshooting

If errors or failures occur the client will always attempt to shut down cleanly, that is stop all running jobs and set their status to "client crashed" and write the last lines of its log output as "launcher output" to each job. This can fail when network connections fail or the client receives a SIGKILL signal causing it to exit immediately. In case of network failures you should still be able to find useful information in the client's logfile on the local filesystem.

5.4 Verifiers

Verifiers are programs that the client runs after a job finishes. Verifiers are getting passed the instance of the job and the solver output as arguments and are supposed to write a newline character followed by a (textual/ASCII) integer result code at the end of their output. The result code should convey some information about the result of the job, for example whether the output of the solver is correct given the problem instance. This code will be written to the database as "result code" while the verifier's exit code will be written as "verifier exit code". Any output the verifier writes to standard out will be written as "verifier output". The call specification for a verifier binary looks like this:

```
./verifier_binary <path_to_instance> <path_to_solver_output>
```

! → We provide a verifier for the SAT problem that works on CNF instances in DIMACS format and solvers that adhere to a certain output format (see the source code). If you want to write an own verifier specific to your problem you can also use the source code as implementation example. Note that you have to make sure that your possible result codes are specified in the *ResultCodes* table in the database before running clients or there will be errors when the client tries to write results. By convention, the web frontend and GUI application consider result codes that begin with a decimal "1" as correct answers.

5.5 Experiment priorization

Sometimes it can be useful to compute several experiments in parallel but give some a higher priority than others. In order to accomplish that, experiments can be marked as inactive and individual jobs can be prioritized. Only jobs of active experiments with priority equal to or greater than 0 are considered for processing by the client. Futhermore, experiments can be assigned a priority. The clients will then try to match the relative number of CPUs working on an experiment with its relative priority to all other experiments that are assigned to the same grid queue. For example, if you have three experiments with priorities 100, 200 and 300 respectively the running clients will try to have 16% of CPUs working on the first, 33% of CPUs working on the second and 50% of CPUs working on the third experiment.

! → The client is running solely on Unix and is not distributed yet for Windows systems.

6 Web Frontend

6.1 Introduction

The Web Frontend provides access to experiment information and analysis tools in a read-only manner and accessible by a web browser.

6.2 System requirements

The web frontend is implemented as Python WSGI web application and makes use of several libraries. Since it interfaces with R to draw plots it also depends on R and a Python interface to R, which unfortunately only works properly on Linux right now. WSGI applications can be deployed on a variety of web servers or even run standalone on a web server that comes with the Python standard library. The following list contains all dependencies and prerequisites of the Web Frontend (see 6.3 for installation instructions).

- Python 2.6.5 or 2.7 http://www.python.org
- R 2.11 (language for statistical computing and graphics)
- R package 'np' (available via R's CRAN)
- SQLAlchemy 0.6.5 (SQL Toolkit and Object Relational Mapper)
- mysql-python 1.2.3c1 (Python MySQL adapter)
- Flask 0.6 (Micro Webframework)
- Flask-WTF 0.3.3 (Flask extension for WTForms)
- Flask-Actions 0.5.2 (Flask extension)
- Werkzeug 0.6.2 (Webframework, Flask dependency)
- Jinja2 2.5 (Template Engine)
- PyLZMA 0.4.2 (Python LZMA SDK bindings)
- rpy2 2.1.4 (Python R interface)
- PIL 1.1.7 (Python Imaging Library)
- Numpy 1.5.1
- pygame 1.9 (Graphics library)

6.3 Installation

To get rpy2 working the GNU linker (ld) has to be able to find libR.so. Add the folder containing libR.so (usually /usr/lib/R/lib) to the ld config: Create a file called R.conf containing the path in the folder /etc/ld.so.conf.d/ and run ldconfig without parameters as root to update. Additionally, you have to install the R package 'np' which provides non-parametric statistical methods. This package can be installed by running "install.packages('np')" within the R interpreter (as root).

The following installation example outlines the step that have to be taken to install the web frontend on Ubuntu 10.04 running on the Apache 2.2.14 web server. For performance reasons (e.g. query latency) the web frontend should run on the same machine that the EDACC database runs on.

Example 7:

1. Install Apache and the WSGI module:

```
apt-get install apache2 libapache2-mod-wsgi
```

2. Copy the web frontend files to /srv/edacc_web/, create an empty error.log file and change their ownership to the Apache user:

```
touch /srv/edacc_web/error.log
chown www-data:www-data -R /srv/edacc_web
```

3. Create an Apache virtual host
(new file at /etc/apache2/sites-available/edacc_web)

```
<VirtualHost *:80>
  ServerAdmin email@email.com
  ServerName foo.server.com

  LimitRequestLine 51200000

  WSGIDaemonProcess edacc processes=1 threads=15
  WSGIScriptAlias / /srv/edacc_web/edacc_web.wsgi

  Alias /static/ /srv/edacc_web/edacc/static/

  <Directory /srv/edacc_web>
    WSGIProcessGroup edacc
    WSGIApplicationGroup %{GLOBAL}
    Order deny,allow
    Allow from all
  </Directory>

  <Directory /srv/edacc_web/edacc/static>
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

4. Install dependencies and create a virtual environment for Python libraries:

```
apt-get install python-pip python-virtualenv python-scipy
apt-get install python-pygame python-imaging
virtualenv /srv/edacc_web/env
apt-get build-dep python-mysqldb
apt-get install r-base
echo "/usr/lib/R/lib" > /etc/ld.so.conf.d/R.config
ldconfig
source /srv/edacc_web/env/bin/activate
pip install mysql-python
pip install rpy2
pip install flask flask-wtf flask-actions
pip install sqlalchemy pylzma numpy
```

5. Install R libraries ("R" launches the R interpreter):

```
R
install.packages('np')
```

6. Create a WSGI file at /srv/edacc_web/edacc_web.wsgi with the following contents:

```
import site, sys, os
site.addsitedir('/srv/edacc_web/env/lib/python2.6/site-packages')
```

```
sys.path.append('/srv/edacc_web')
sys.path.append('/srv/edacc_web/edacc')
os.environ['PYTHON_EGG_CACHE'] = '/tmp'
sys.stdout = sys.stderr
from edacc.web import app as application
```

7. Configure the web frontend by editing /srv/edacc_web/edacc/config.py, see 6.4 for details.
8. Enable the Apache virtual host created earlier:

```
a2ensite edacc_web
service apache2 restart
```

9. The web frontend should now be running under http://foo.server.com/

6.4 Configuration

All configuration is done in a Python file located at "edacc/config.py". The options are documented in the sample configuration file which is included in the distribution package. Please read through the configuration options and modify those marked as important. Most importantly, you should disable debugging mode and change the secret key when making the Web Frontend accessible from the network to avoid security problems. Logging is also quite useful to make it easier to find the cause of bugs in the application. At the end of the file you can configure the database connection and the list of EDACC databases that should be made available by the Web Frontend.

! →

Database configuration

6.5 Troubleshooting

When there are errors or bugs and you have set up the Web Frontend under Apache as described in the installation section, Apache will display an "Internal Server Error" page. If you have configured logging, the application will write tracebacks to the logging file. If you haven't set up logging, these tracebacks will end up in Apache's error.log file.

6.6 Features

This section gives a short overview of the features of the Web Frontend. Most features should be self-explanatory or have some additional documentation on the pages themself.

The Web Frontend was designed as monitoring and analysis tools of experiments that are created with the GUI application. Once you have set up some databases and added them to the configuration file of the Web Frontend, the top level page will allow the user to select one of the databases. This leads to a page that displays all experiments of the chosen database and some basic information about their creation date, number of solvers, instances and jobs.

Experiments

An experiment page displays further information about the experiment, such as the number of total, running and crashed jobs. If the experiment is currently being computed, an estimation of the time of completion is displayed next to "ETA".

Monitor progress

Under "Progress" a colored bar visualizes the computation progress. Green color corresponds to finished jobs, red to crashed jobs and orange to jobs that are currently being computed. The links following the progress bar lead to information and analysis pages.

Job browser

The progress page provides a job browser similar to the GUI application. It allows to sort, filter for certain words, show and hide specific columns and download currently displayed data in CSV (comma-separated values)

Export data

format. Displaying more than 1000 results at once can become rather slow, since the browser's Javascript engine has to do a lot of processing to color and format the table.

The solver configurations and instances pages show the solver configurations and instances that are part of the experiment. The instances page

Download instances

provides a download link for all instance files in a tarball. Instances are shown in a table by name, MD5 checksum and their properties (TODO reference), if there are any. The instance name links to a page that displays the first and last few characters of the instance file and provides a download link.

View results

The result pages display the results of the jobs in various formats. "Unsolved instances" and "Solved instances" list the instances that were not solved by any solver or solved by at least one solver respectively. "By solver configuration" leads to a page, where after selecting a solver configuration that is part of the experiment a table is displayed containing all the results of the jobs of the solver configuration by instance and run number. "By instance" leads to a page, where after selecting an instance all results obtained on this instance are displayed by solver configuration

Export results

and run number. All tables can be exported (i.e. downloaded) in CSV format.

Analyse results

Analysis pages provide various plots of results and statistical tools such as correlation and probabilistic domination calculations. All plots can be directly saved in PNG format as they are displayed in the browser

Export plots

or downloaded in PDF or EPS format. For some plots the application generates an R script which can be adjusted as neccessary. Most plots allow to download the underlying data in CSV format.

Some plots allow the selection of multiple instances. In this case, you can use a filter to narrow the selection of the instances listed. Please refer to the example which is displayed next to the filter text field.

Ranking of solver configurations

The ranking is determined by the number of successful runs but the ranking table can be sorted by any of the displayed measures.

Index