

MooN

User Manual

MooN - A Framework for Metaheuristic Optimization.
© 2003-04 PG 431, Chair of Systems Analysis, University of Dortmund.
MooN is licensed under the GNU General Public License.

Preface

From the beginning of the development of MooN we had ambitious plans to not only fulfill the requirements of the university course, but to provide a tool to the scientific community with practical value. There were only a few comparable projects which aimed at making the research work on evolutionary algorithms easier, and which was what we identified as our main goal in the early stages of the project. We wanted to help scientists focus on the 'real work', giving them the possibility to mostly neglect the 'craftmanship' of programming an environment for the algorithms and problems under examination. We hope that our concept of the plug-in structure lives up to the ambitions we had in mind. Anyway, we already found the design quite helpful during the last stages of our project. We hope that some people will find that as well.

Projektgruppe 431

The 'PG 431' is:

Selcuk Balci, Sören Blom, Daniel Blum, Vedran Divkovic, Dirk Hoppe, Djamila Lindemann, Ulf Schneider, Bianca Selzam, Thomas Tometzki, Marko Tomic, Igor Vatulkin and Stefan Walter.

Contents

Preface	ii
1 Introduction	1
1.1 Requirements	2
1.2 Installing Moon	3
2 Managing Plug-Ins	4
2.1 Installing Plug-Ins	4
2.2 Deleting Plug-Ins	5
3 Managing Single Runs	6
3.1 Problem	7
3.2 Heuristic	9
3.3 Exit Condition	11
3.4 General Options	12
4 Managing Complete Runs	14
4.1 Composing	14
4.2 Running	15
4.3 Saving	16
4.4 Loading	16
5 Command Line Usage	17
5.1 Invoking Moon from the Command Line	17
5.2 (Un-)Installing Plug-Ins from the Command Line	18
5.2.1 Installing a Plug-In	18
5.2.2 Removing a Plug-In	18
5.2.3 Listing Plug-Ins	18

1 Introduction

"One small step for man, one giant leap for mankind."
- Neil Armstrong on landing on the moon

Welcome to MooN!

This is a short user manual which should help you exploring the MooN system. MooN is a free software tool that provides a plug-in-based framework for implementing heuristics and optimization problems. These plugins can be installed into the core system, after which they are immediately available to be used for experiments. The MooN core system provides all the functionality to run interesting experiments on the installed plugins. There are a number of plugins included in the MooN distribution as it can be downloaded from <http://ls11-www.cs.uni-dortmund.de/lehre/SoSe03/PG431/MooN/moon.htm>.

In the problem section you can find common test functions like:

- spherical function
- Schwefel function
- Griewank function
- Travelling Salesperson problem

In the heuristic section there are implementations of:

- genetic algorithms
- particle swarm optimization
- ant colony optimization
- evolution strategy

and other heuristics.

You can also find a number of exit conditions, such as

- number of generations
- exit at a given time
- fitness threshold

MooN has been developed by students of the University of Dortmund, Germany. It was a project work of a course called 'Metaheuristics - New Ideas for Optimization'. We wish you a lot of fun with MooN and hope that it makes your scientific work easier!

1.1 Requirements

MooN is written in the Java programming language. An installation of the Java 2 Platform Standard Edition (J2SE), version 1.4.2¹ is needed. For details about J2SE downloading and installing instructions refer to:

<http://java.sun.com/j2se/1.4.2/index.jsp>

Additionally some third party packages are needed to run MooN. Those packages are **Log4J**, **Xerces** and **JDOM**.

Log4J, version 1.2.8

Log4J is an *Apache Software Foundation* project for advanced logging control. The main benefit is the logging output can be controlled through a configuration file without modifying the source code. Log4J can be obtained from the project's homepage:

<http://logging.apache.org/log4j/docs/>

Xerces 2 for Java, version 2.6.x

Xerces is a highly modular XML parser, both for XML parsing and generation. MooN makes extensive use of Xerces, as XML is a main feature for providing extensibility and universality. Xerces is also an Apache Software Foundation project, available on their XML website:

<http://xml.apache.org/xerces2-j/index.html>

Note that the Xerces2 *Java* Parser is needed. Two libraries out of the binary distribution package are of interest: `xercesImpl.jar` and `xml-apis.jar`.

JDOM, version 1.0 beta 9

JDOM is a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code. It is available on the project's homepage:

<http://www.jdom.org/>

¹Generally the latest installation of J2SE and the third-party packages should work with MooN. Since downward compability is not always achieved, the exact version numbers are provided as used to develop MooN

1.2 Installing MooN

MooN is distributed as a `jar`-package including an installation routine. Typing

```
java -jar install.jar
```

at the command line will start the installation process. Depending on the systems configuration double-clicking the `install.jar` file will do the same. The graphical installation manager that is started by this is self-explanatory.

Files and Directory Structure Depending on the decisions during installation the following subdirectories will be created in the MooN directory:

- `bin` - All class files and libraries are placed here. Also, necessary third party libraries have to be copied to `bin/lib` (see below).
- `doc` - The MooN Documentation: user and developers manual, readme and the JavaDoc can be found here.
- `misc` - The `jar` files of the included plugins and a sample R script for statistical analysis reside here.
- `src` - The source files of MooN.
- `Uninstaller` - The uninstaller for automatical removing of MooN.

Copying the necessary Libraries In order to run MooN some third party libraries must be copied to the `moon/bin/lib` directory. For details about those libraries and where they can be obtained refer to section [1.1](#).

After downloading the described libraries, they need to be copied to `moon/bin/lib`. MooN can then be started by either selecting the `Moon.bat` or `Moon.sh` depending on the underlying operating system.

2 Managing Plug-Ins

"Computers are useless. They can only give you answers."
- Pablo Picasso

One of the main advantages of MooN is the plug-in approach. Through this approach one can install and remove heuristics, problems and exit conditions at any time. Given a valid heuristic plug-in (received through a colleagues email, downloaded from the Internet, etc.), installing it into your installation of MooN is a matter of seconds. The steps needed are described in detail below.

2.1 Installing Plug-Ins

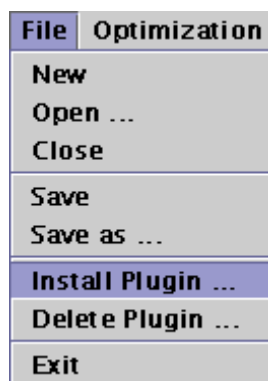


Figure 1: Installing a plug-in

Installing a plug-in is as simple as loading a file: In the File menu choose **Install Plug-In** (figure 1). In the appearing dialog, search and choose your plug-in `.jar`-file and click **Open** (figure 2). The plug-in will be stored in the default plug-in directory, ready for use and should appear in the list during the configuration of a single run (see section 3.2).

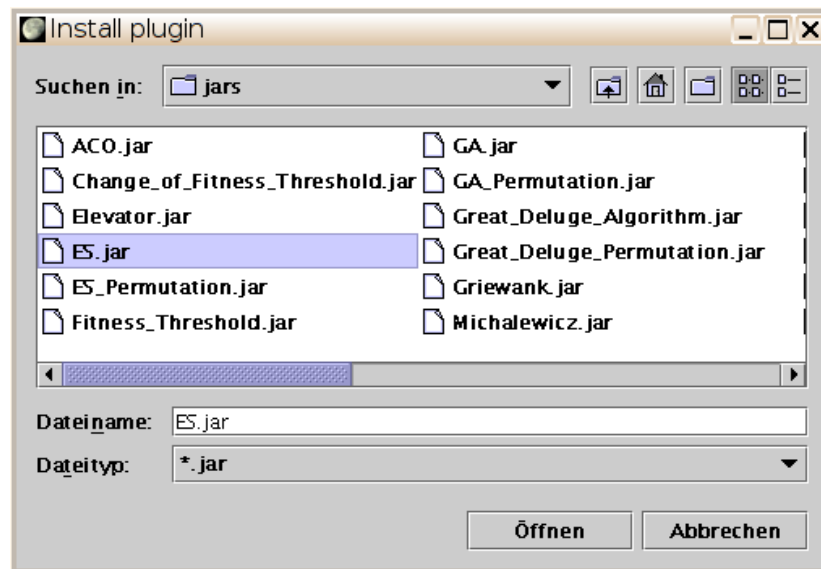


Figure 2: Selecting a plug-in

2.2 Deleting Plug-Ins

To remove a plug-in choose the Delete plug-in entry from the File menu (figure 3). From the list of installed plug-ins choose the one to remove and confirm this by clicking OK (figure 4). The plug-in files are deleted from the plug-in directory.

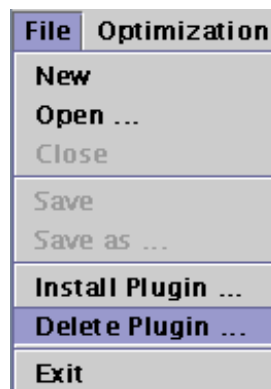


Figure 3: Deleting a plug-in

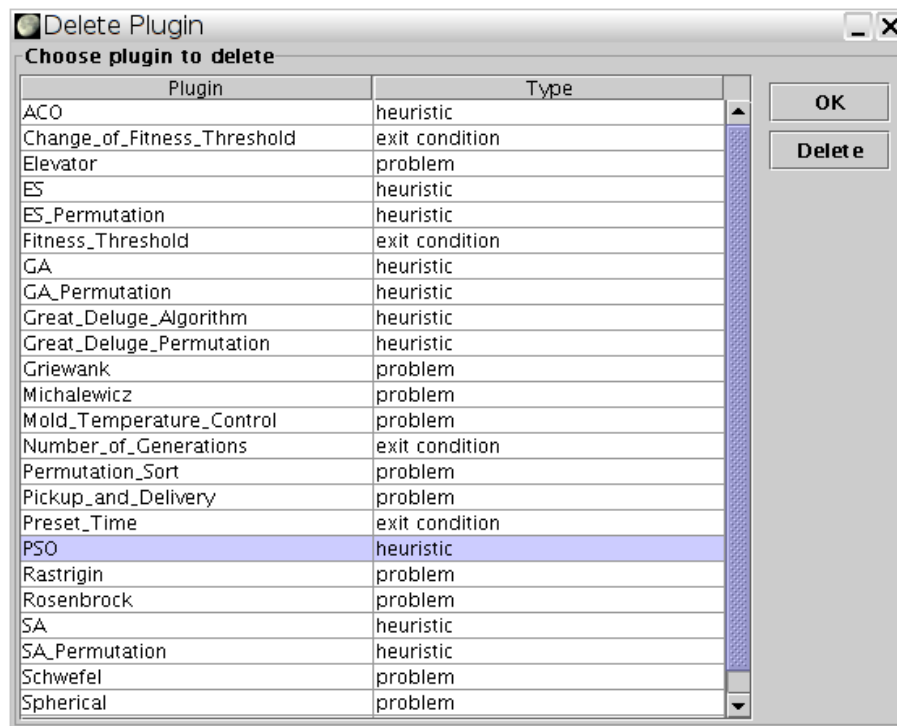


Figure 4: Selecting a plug-in for deletion

3 Managing Single Runs

"Run, Forrest, run!"
- Jenny Curran to Forrest Gump

The smallest execution unit in MooN is called *single run*. A single run represents one experiment with its parameter options for the respective heuristic, problem and exit condition. In order to create a single run in MooN one needs to pick at the least one heuristic, one problem and one exit condition.

The single run can be started with its default parameters. However, if real life experiments are to be conducted, the options of the three plug-ins will need to be tuned. After all options are set correctly the Accept button is enabled.

3.1 Problem

We start with an explanation of the problem and its parameter configuration.

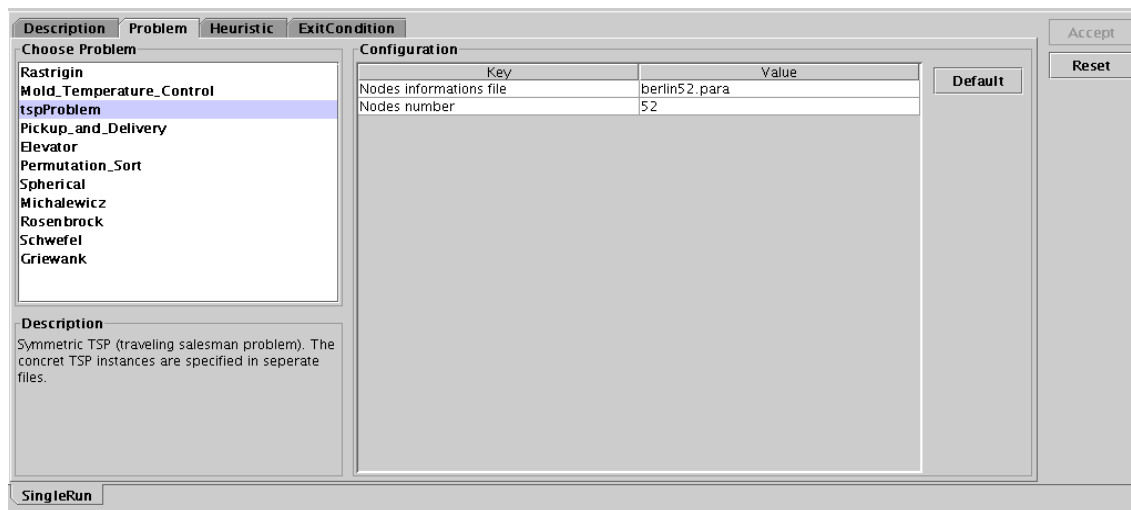


Figure 5: Single run parameter dialog

The single run problem dialog (close-up shown in figure 5) enables the user to select problem specific options.

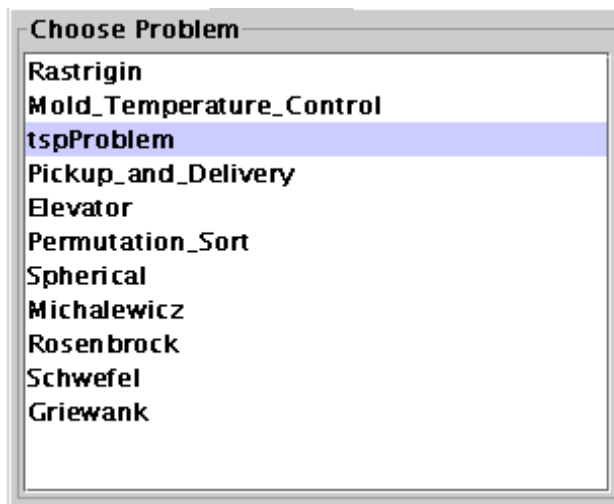


Figure 6: Problem selection dialog

To the left part of figure 5 there is a list with all installed problem plug-ins (close-up in figure 6). If a desired plug-in is not found in the list, it needs to be installed. After installing it (see section 2.1) it will be visible in this list and can be used in single runs from there on.

When a problem is selected from the list, a detailed description of the plug-in is displayed in the lower left corner, shown in figure 7.

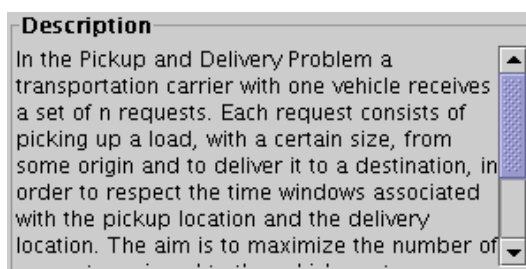


Figure 7: Problem description dialog

Key	Value
Nodes informations file	berlin52.para
Nodes number	52

Figure 8: Problem configuration dialog

After deciding which problem to optimize within this single run, ensure that the problem is configured to the needs of the experiment it will be take part in. The problem configurator as shown in figure 8 shows all available options with their values for the selected problem. To change the value for a specific option simply select the corresponding table cell by clicking it and editing the value.

If it becomes necessary to reset values, clicking the **Default** button resets *all* parameter values to their factory setting. The button is located to the right of the parameter list (see figure 5).

3.2 Heuristic

In this section the heuristic part of the single run configuration is described. All parameters regarding the strategy of the heuristic can be set as described here. The single run heuristic dialog is similar to the problem dialog. This is where all runtime options of the heuristic can be set.

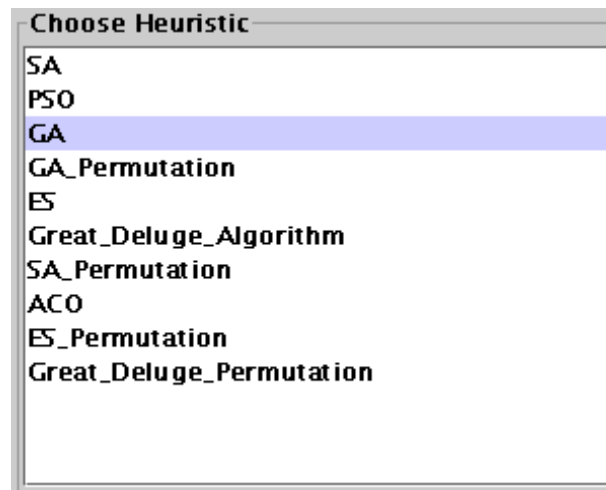


Figure 9: Heuristic selection dialog

To the left part of figure 5 a list with all installed heuristic plug-ins can be found (figure 9).

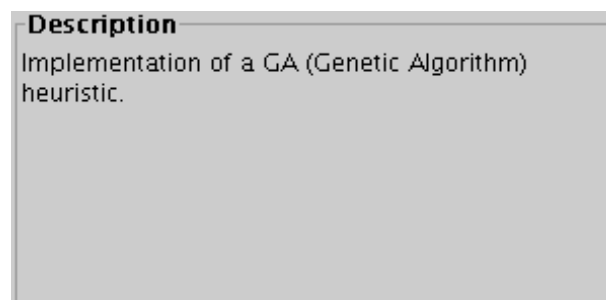


Figure 10: Heuristic description dialog

When a heuristic plug-in is selected, a detailed description of its function is displayed in the lower left corner as shown in figure 10.

Similar to the situation in section 3.1 the list shown in figure 11 contains all available options for the selected heuristic. Editing values in the list is done in the same way as described there and the **Default** button has the same effect as well.

In addition to the dialog elements in the problem section (3.1) there are *output handler* options. The output handler is the part of MooN that writes experiment data to your hard disk for later evaluation. Since different heuristics may have

Configuration	
Key	Value
Population size	100
Solution minimum	-10.0
Dimension	10
Selection size	10
Maximal mutationincrement	0.1
Probability of crossover	0.8
Probability of mutation	0.1
Solution maximum	10.0

Figure 11: Heuristic configuration dialog

different data to examine you can parameterize the output of the single run. There are two important settings to consider: the output interval and the output file. Depending how fine grained the heuristics output should be on a specific parameter, the logging interval takes any positive integer including zero. A value of zero means that no logging one that parameter is done, a positive value n means that logging for this parameter is done every n -th generation. When considered usefull for later data analysis, the output for every parameter can be written into individual files. This option is enabled by activating the **Log categories to separate files** switch (see figure 12).

OutputHandler	
Log filename:	<input type="text" value="Default.log"/>
<input type="checkbox"/> Log categories to separate files	<input type="button" value="Default"/>
Category	Logging interval
global_best_individual	1
current_best_individual	1

Figure 12: Heuristic output handler dialog

3.3 Exit Condition

The exit condition is the part of MooN that controls the termination of a single run. For example, a single run can be stopped after a specific time or after a certain number of generations. Exit conditions are also realized as plug-ins. One exit condition needs to be selected for a single run to be complete.

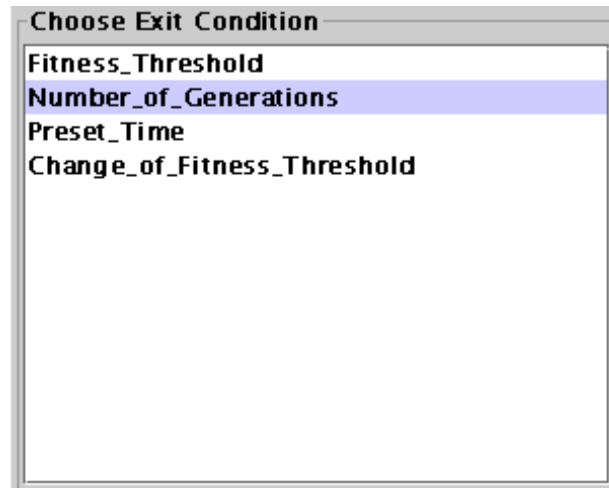


Figure 13: Exit condition choose dialog

The dialog shown in figure 13 is used to select the exit condition. All installed exit condition plug-ins are listed. When selected, a detailed description of the plug-in is displayed in the lower left corner as shown in figure 14.

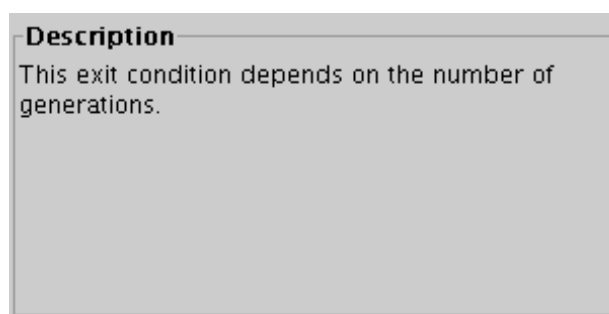


Figure 14: Exit condition description dialog

As before, parameter values of the selected exit condition can be edited in the associated list.

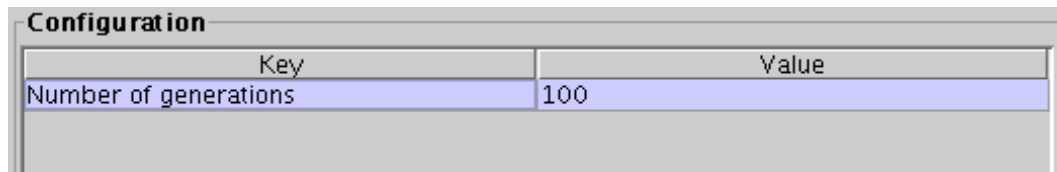


Figure 15: Exit condition configuration dialog

3.4 General Options

Single Run Description

In the description section of a single run one can enter a detailed description of a single run. It is recommended to put a single, one line short description in the first line since it will be displayed in the complete run overview. The length of the description is not limited and the visible area provides enough space for detailed information regarding a single run (figure 16).

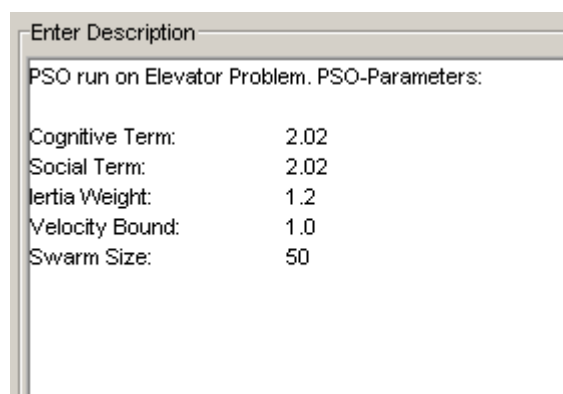


Figure 16: Single run description dialog

Although descriptions are not necessary for the execution of the single run, it is strongly recommended to explain the experiment here in order to be able to understand the purpose of the configuration without reviewing all values again.

Repeating Single Runs

Another feature, designed to conduct real life experiments easily, is the ability to repeat a single run several times by changing the repetitions' value in the dialog shown in figure 17. This reduces the variance and generates statistically more significant results without creating new single runs with identical parameter settings.

Copying Single Runs

Another handy feature is the Copy item from the Single run menu. It is helpful for creating complete runs that consist of *similar* single runs. When applied to a

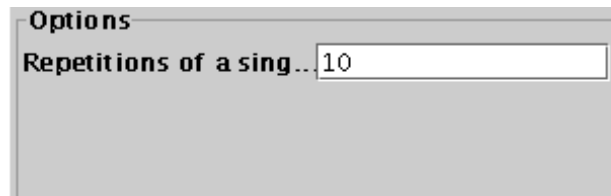


Figure 17: Single run repetitions dialog

selected single run in the complete run list, the copied single run is appended to the list and the desired changes can be applied to it.

4 Managing Complete Runs

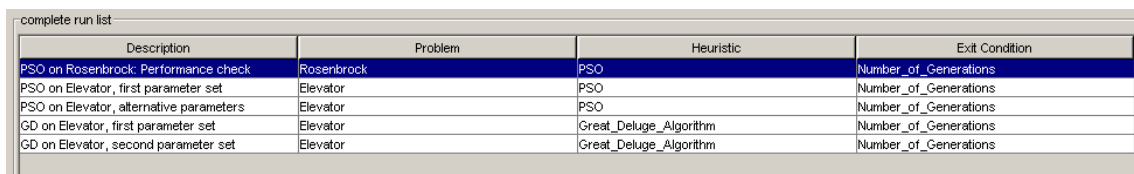
”There is a theory which states that if ever anybody discovers exactly what the universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.”

- Douglas Adams

In this part managing complete runs is explained. There are four main activities with regard to complete runs: *composing*, *running*, *saving* and *loading* complete runs.

4.1 Composing

Single runs are administered in a task list called the 'complete run list' (figure 18). Every line represents a single run, hence the whole list represents the complete run. To create a new complete run, select **New** in the **File** menu or click on the icon for a new complete run in the toolbar.



Description	Problem	Heuristic	Exit Condition
PSO on Rosenbrock: Performance check	Rosenbrock	PSO	Number_of_Generations
PSO on Elevator, first parameter set	Elevator	PSO	Number_of_Generations
PSO on Elevator, alternative parameters	Elevator	PSO	Number_of_Generations
GD on Elevator, first parameter set	Elevator	Great_Deluge_Algorithm	Number_of_Generations
GD on Elevator, second parameter set	Elevator	Great_Deluge_Algorithm	Number_of_Generations

Figure 18: Complete run list

Adding a single run to the list To add a new single run, select **Add** in the **Single run** menu or click on the "plus" button.

Duplicating single runs A feature which is very handy when composing list of very similar single runs is the copy feature. To duplicate an existing single run, select it and chose **Copy** from the **Single run** menu. The clone will appear at the end of the list.

Changing the order of execution To move a single up and down the list, one can use the **Move up / Move down** in the **Single run** menu or the arrow buttons in the toolbar.

Removing single runs The "minus" button or the **Delete** entry in the **Single run** menu will remove a selected single run.

4.2 Running

After a complete run has been loaded or newly created, it can be started by selecting **Optimization/Run** from the menubar. Alternatively, the "play" button in the toolbar (figure 20) does the same. Depending on how complex the complete run is, the running process can take a long or short time. However, the course of optimizations can be followed on the *Runtime Visualization Graphic* that opens up automatically upon a run as shown in figure 19.

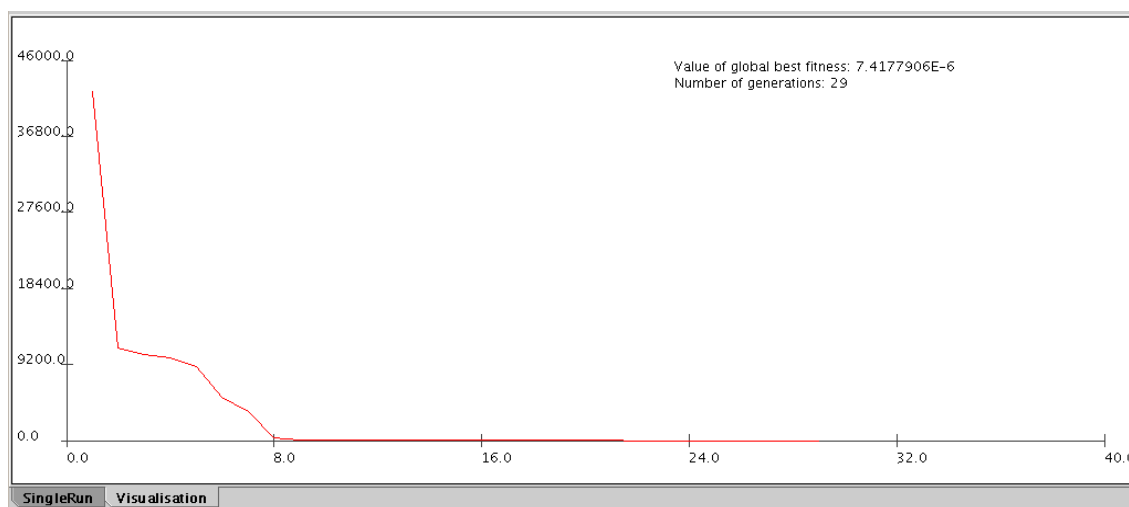


Figure 19: Runtime visualization

If you want to observe an interesting state of the optimization in more detail the current run can be paused by the "pause" button in the toolbar (figure 20). In case you made a mistake in the configuration of a very long run you can cancel the complete run by pressing the "stop" button.



Figure 20: The toolbar

4.3 Saving

In order to save a complete run choose the **Save** entry from the file menu (see figure 21) or press the disk symbol on the toolbar (see figure 20). When saving a complete run for the first time MooN will ask for a file name. Otherwise the file on the disk is replaced with the recent complete run. Saving an already saved run into a different file is done by selecting **Save As** from the File menu.



File Optimization Single run Help

Figure 21: The menubar

After entering a proper name, pressing the **Ok** button of the File Save dialog and MooN writes the complete run to the hard disk.

4.4 Loading

Complete runs are saved as `.xml` configuration files. To load a saved complete run choose **Open** from the File menu. This opens a file chooser similar to the one in figure 2. After selecting the desired file, pressing the **Ok** button will load it. Note that loading might not be successful if the file specifies plug-ins that are not installed or have different parameters. This might be the case if the complete run was created with an installation of MooN that had more or other plug-ins installed. Installing the missing plug-ins will help this problem.

5 Command Line Usage

"Bad Command or File Name. Good try, though."
- Anonymous

For automation purposes MooN can be invoked from the command line. This can be very helpful to conduct complex and time consuming experiments on environments that only allow shell access, e.g. batch systems or mainframes. Following operations can be called from the command line:

- starting complete runs
- installing plug-ins
- uninstalling plug-ins
- listing plug-ins

These functionalities are described in detail in the following subsections.

5.1 Invoking MooN from the Command Line

Starting MooN from the command line is simple. All one needs is to specify the path to a valid XML complete run definition. This XML file can be created using the GUI (as described in the preceding chapters). Of course, manipulating it with a text editor is also possible (see the developers manual for details on the file structure). One can call Moon with the parameter `-r` (for run):

```
Moon -r <pathToCompleteRunXML>
```

MooN will process the tasks in the XML definition file. There will be no runtime visualization, only the described output handling is performed.

Even if no batch system is used, one might consider running MooN from the command line, since avoiding the runtime visualization causes a noticeable performance gain.

5.2 (Un-)Installing Plug-Ins from the Command Line

As mentioned above, one can install, remove and list the plugins.

5.2.1 Installing a Plug-In

One needs a valid plug-in in form of a `.jar` archive. Install the plug-in by using the `-i` parameter:

```
Moon -i <pathToPlugin.jar>
```

The plug-in will be extracted and stored in the standard plug-in directory of MooN (`./plugins` in the Moon directory). Installing it into a different directory is done by using the optional `-d` parameter:

```
Moon -i <pathToPlugin.jar> -d <pluginDirectory>
```

5.2.2 Removing a Plug-In

Removing a plug-in requires its full name. To obtain a list of all plug-ins from the command line see [5.2.3](#). Knowing the name, uninstalling a plug-in is done with the `-u` switch:

```
Moon -u <pluginName>
```

5.2.3 Listing Plug-Ins

The `-l` switch is used for listing all installed plug-ins:

```
Moon -l
```

We hope that this manual will be found helpful in introducing the user to MooN and its usage.