DIXER User Manual

for version 2.0.1

Rafał Latkowski
rlatkows@mimuw.edu.pl
http://logic.mimuw.edu.pl/~rses/dixer/

Contents

Chapter 1

Introduction

1.1 DIXER

DIXER (DIstributed eXEcutoR) is a part of *Rough Set Exploration System* (RSES, [2]). DIXER is designed to help experimenters that would like to run their experiments on many computers. DIXER allows to employ computers connected with an intranet or internet to distribute experiments on them automatically. It creates a virtual cluster of computers for this particular purpose. DIXER takes care on all aspects of scheduling, communication and data transfer, so it requires no additional network services (like e.g. ftp or nfs). It is mainly designed for experiments based on RSES-Lib software, but as an open architecture allows to plug-in new modules without modification of the source code. DIXER consist of two applications: Server and Client. This manual descSribes how to use *DIXER Server*, *DIXER Client* and also plugable modules *Branch'n'Bound* and *RSParallel*. See also DIXER Module Developers Guide.

1.2 DIXER Server

DIXER consist of two applications: Server and Client. *DIXER Client* should be run on all computers that should cooperate in distributed computations. *DIXER Server* should be run only once, on administrator's computer. DIXER Server schedules experiments (jobs) for all executed DIXER Clients. It contains a graphical user interface that allows managing whole DIXER Cluster, (see fig. 1.1). To clarify some concepts and make this manual more precise we specify here distinction between jobs, taks and types.

Task type. At the DIXER Server runtime there are fixed number of task types. They can be more or less identified with installed plugable modules[1]. Task types are configured in a configuration file named `dixer.conf` which is described in detail in section 3.3.

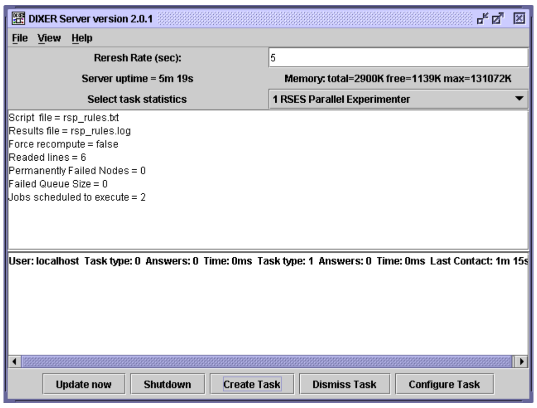
Task. Tasks are created and dismissed dynamically by a user during runtime. User can create a new task of any previously specified task type. There can by more than one task of a particular task type. The computation time of each task vary a lot and can take from a couple of minutes up to weeks (usually 1–100 hours). Each plugable module implements[2] a mechanism to restart from the interruption point and to do not repeat already computed jobs. This is extremely important in a case of e.g. power or computer failure.

Job. Created tasks are non-elementary beings and consist of jobs that can be interpreted as elementary operations on DIXER Cluster. These jobs are sent to client and are visualized at the DIXER Client GUI. The computation time of each job vary and can take from a couple of seconds up to about two hours (usually about five minutes). If there is any failure during job evaluation, the results are dropped and the job returns to queue of scheduled jobs[3].

[1]A plugable module can be specified as a task type multiply times. They will be identified by different names (labels) (c.f. section 3.3), but they will share all module specific configuration parameters.

[2]It is at least true for all modules that are distributed with the DIXER software.

[3]It returns only a specified, finite number of times to avoid permanent blocking of jobs queue

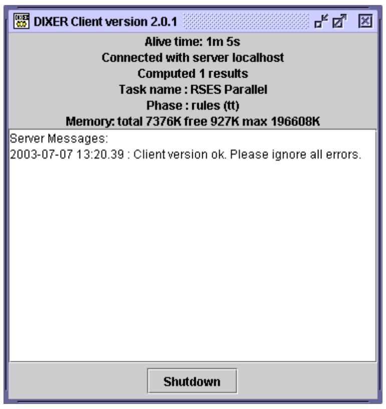


Reresh Rate (sec): 5
Server uptime = 5m 19s
Memory: total=2900K free=1139K max=131072K
Select task statistics: 1 RSES Parallel Experimenter

```
Script  file = rsp_rules.txt
Results file = rsp_rules.log
Force recompute = false
Readed lines = 6
Permanently Failed Nodes = 0
Failed Queue Size = 0
Jobs scheduled to execute = 2
```

```
User: localhost  Task type: 0  Answers: 0  Time: 0ms  Task type: 1  Answers: 0  Time: 0ms  Last Contact: 1m 15s
```

Figure 1.1: DIXER Server



```
Alive time: 1m 5s
Connected with server localhost
Computed 1 results
Task name : RSES Parallel
Phase : rules (tt)
Memory: total 7376K free 927K max 196608K
```

```
Server Messages:
2003-07-07 13:20.39 : Client version ok. Please ignore all errors.
```

Figure 1.2: DIXER Client

Usually, DIXER Server consumes a little of system resources, both CPU time and system memory. Exact values, however, depend on current work. DIXER Server can also require some additional files on disc that are used by plugable modules as data files needed in experiments.

1.3 DIXER Client

DIXER consist of two applications: Server and Client. *DIXER Client* should be run on all computers that should cooperate in distributed computations. *DIXER Server* should be run only once on administrator's computer. DIXER Client executed all experiments scheduled by DIXER Server. It contains simply graphical user interface on which one can supervise a program status (see fig. 1.2). On demand this GUI can be disabled and then DIXER Client runs completely silent. Usually, DIXER Client consumes a lot of system resources, both CPU time and system memory. Exact values, however, depend on current work that is scheduled by DIXER Server. DIXER Client can also require some disc space in order to store data files needed in experiments.

Chapter 2

Installation

2.1 Setup program

The DIXER Software is provided for continency with a MS-Windows setup program (see fig. 2.1). This program simplifies a lot the installation procedure. In few mouse clicks user can install the full DIXER Software package, this documentation, sample files and shortcut icons that executes the DIXER Server and Client. For a special purposes also similar setup program is provided for separately: DIXER Server and DIXER Client. The DIXER Client package is sufficient for the workstations that will be used only as a working elements of the DIXER Cluster. The DIXER Server package contains this documentation and files necessary to run the DIXER Server and manage the cluster of working elements.

2.2 Package content

The full package of the DIXER Software contains:

1. This DIXER User Manual,
2. `dixer.jar`— the binary archive with the DIXER Software,
3. RS-Lib — the computational core of the Rough Sets, Exploration System (RSES)
4. Examples for provided plugable modules,
5. Set of shortcuts and batch scripts for executing different parts of the DIXER Software.

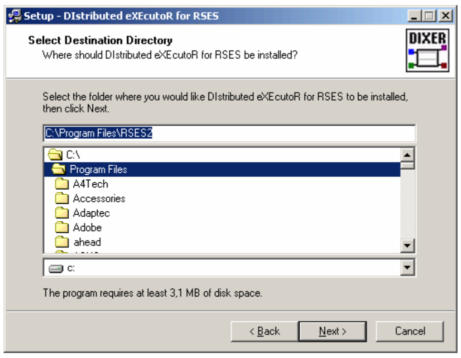
The DIXER Server package does not contain the RS-Lib computational core and scripts for the DIXER Client part. The DIXER Client package does not contain this manual, scripts and examples for the DIXER Server.

The batch script files allows to customize the execution of the DIXER software. The most important is the customization of the available memory for the DIXER Client (see files `client.bat` and `runclient.bat` that executes the DIXER Client in an infinite loop).

The files `rsp_rules.txt` and `rsp_decomptree.txt` in the directory `server` are the example scripts for RSParallel plugable module. The files in directory `server/data` contains a data files for these examples and also an example file for the Branch'n'Bound plugable module. The default settings of these plugable modules make a use of provided example files.

2.3 Java Runtime Environment

To properly run any of DIXER software components a Java Runtime Environment in version at least 1.4 is required (c.f. [5, 6]). The Java Runtime Environment (JRE) is included also in any Java Software Development Kit (JDK or J2SE_SDK). The JRE or JDK can be downloaded directly from


Setup - DIstributed eXEcutoR for RSES

Select Destination Directory

Where should DIstributed eXEcutoR for RSES be installed?

Select the folder where you would like DIstributed eXEcutoR for RSES to be installed, then click Next.

C:\Program Files\RSES2

C:\
Program Files
A4Tech
Accessories
Adaptec
Adobe
ahead

c:

The program requires at least 3,1 MB of disk space.

Figure 2.1: DIXER Setup program.

`http:\\java.sun.com\`. The DIXER Software was deeply tested with various versions of Sun's JRE 1.4 and should operate also on any new version of Java Virtual Machine (c.f. [4]). The DIXER Software does not work with Java Runtime Environment 1.0, 1.1, 1.2 and 1.3.

2.4 Executing

The provided `dixer.jar`archive contains all necessary files to execute the DIXER Software. The execution of the parts of the DIXER Software can proceed as follows:

1. by selecting a program shortcut in the Start/Programs menu or at the user desktop,
2. by executing a batch script from the program directory,
3. by executing Java Virtual Machine with special options.

The special parameters mentioned above should be analogous to the provided here examples:

1. `java -cp ../lib/dixer.jarrslib.dixer.client.Server` — for executing the DIXER Server,
2. `java -Xmx512M -cp ../lib/dixer.jar -Djava.library.path=../lib/ rslib.dixer.client.Client` — for executing the DIXER Client,
3. `java -cp ../lib/dixer.jarrslib.dixer.client.ClientConfigurator` — for executing the DIXER Client Configurator.

For details and explanation of the above parameters see the Java Tools Documentation [6].

Chapter 3

Configuration

3.1 Server Configuration

The configuration of the DIXER Server is stored in a configuration file that can be shared with DIXER Client. This configuration file concern with advanced properties of server itself and plugable modules and can be leaved for experienced users or an administrator. A default version of this file is attached to each distribution of DIXER software, so there is no need (especially for beginners) to alter mentioned configuration file.

3.2 Client Configuration

The configuration of the DIXER Client consist of two stages. The first stage concern with selection of server in an intra- or internet. Such a configuration is in scope of interest of all users. The second stage concern with advanced properties of client itself and plugable modules and can be leaved for experienced users or an administrator. The first stage of configuration can be made by a graphical user interface. The advanced properties are accessible in textual configuration file. A default version of this file is attached to each distribution of DIXER software, so there is no need (especially for beginners) to alter mentioned configuration file.

3.2.1 Server address

DIXER Client requires an address of DIXER Server that will schedule jobs for remote execution. One cannot execute the DIXER Client without specifying a server to connect to. User can define server name or address[1] in two ways:

1. as a command line parameter to client class[2],
2. as a file named `dixer.connection` in current directory.

If the command line parameter is specified than it will be used first. If client class is started without parameter then there will be an attempt to read settings from `dixer.connection` file in current directory. If this also fails than program will ask about server name or address by showing a ClientConfiguration window (see fig. 3.1).

DIXER Client Configurator can be executed as a standalone application by running `rslib.dixer.client.ClientConfigurator` class. When a client can not connect with a server (cf. fig. 5.1) it does not necessarily means that configuration is wrong. It may also happen that server is shut down or that there are some network problems (e.g. broken cable).

[1]IP address is required or any other name that can be resolved by `gethostbyname`.
[2]`rslib.dixer.client.Client`, see chapter 5 for details.

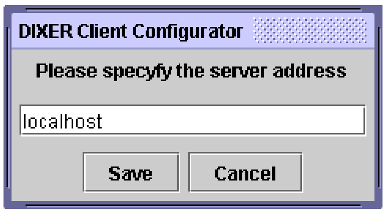


Figure 3.1: DIXER Client Configurator window.

If `dixer.connection` file is already written (e.g. by the Client Configurator) than it will be implicitly used without notification. If user want to change the server address (name) it is necessary to delete the `dixer.connection` file in current path. If DIXER Client is executed without additional command line parameters than it will prompt about new server address or name.

3.3 Configuration file

Configuration file contains advanced parameters for the DIXER Sever and Client. The name of the configuration file should be passed to the main server class (i.e. `rslib.dixer.client.Server`) as a command line parameter (c.f. 4.1). If such a parameter is omitted than the standard name "`dixer.conf`" is used. The DIXER Server searches for the specified file in available class paths (c.f. ClassLoader.getResourceAsStream() in Java API Documentation [5]). If a resource of a specified name cannot be found than the server searches for this file in the current directory. It is safe and recommended to use the same configuration file for both server and client, especially when they share the same version of code. It is not recommended to share configuration file across the different versions of code (e.g. when the set of accompanying modules is changed).

The DIXER Client uses the same method for reading the configuration file and for passing the name of this file (c.f. 5.1).

In the configuration file each parameter is described in following way:

$$< parameter > = < value >$$

Whitespace characters before and after equation sign are not important[3]. Empty lines and lines that begins with "#" character are ignored. Below the description of each parameter is provided.

3.3.1 Connection

- `version` (mandatory, for server and client). Specifies version number of software. Usually modified only by a developer that made changes in source code. No default value. Set to 2.0.1.
- `server.port` (mandatory, for server and client). IP port number on which server listen for client. No default value. Set to 7333. If there are any problems because of network firewall or user privileges than one can try to modify this value e.g. to port below 100, to port above 4096 etc.[4]
- `server.clienttimeout` (mandatory, for server only). Time (in ms) that server waits for an answer from a client. After that time server assumes that client failed and closes communication channel. This time should be longer than execution time of any job on any computer that will be used during experiments. Some times it is reasonable to set this value not too large, especially when we have some computers with small amount of memory (due to trashing, i.e. running out of the physical memory). The server attempts to shutdown client before closing connection by sending a "`kill`" command. No default value. Set to 3600000 (1 hour).

[3]See java.util.Properties in Java API Documentation for details [5].
[4]See respective documentation or manual of used operating system.

- `server.waittimeout` (mandatory, for server only). Maximal time (in ms) that server will sleep when there are no more jobs to send. This time This time should be shorter than `client.server-timeout` time. No default value. Set to 240000 (4 minutes).

- `client.querytimeout` (mandatory, for client only). Time (in ms) that client sleeps after unsuccessful attempt to connect with a server. No default value. Set to 10000 (10 seconds). After that time client will attempt to reconnect with a specified server.

- `client.servertimeout` (mandatory, for client only). Maximal time (in ms) that client will wait for answer from a server. If server is overloaded or network (physical) connection is temporarily broken answer could be delayed. No default value. Set to 300000 (5 minutes).

- `server.buffersize` (mandatory, for server only). Size in bytes of data blocks transferred at once during "`filetransfer`" job. No default value. Set to 1024.

- `hostname.<HOSTNAME>` (optional, for server only). Creates a dictionary of IP addresses. Each client that contacts to the host is recognized by IP address. If the searching for name fails[5] this client will be displayed as its IP address. To avoid this problem user can create its own dictionary of hosts that overrides any other method of assigning a name. The value of this parameter should be a IP address in textual form "X.X.X.X". No default value. Some entries can be already written at the bottom of the default configuration file.

3.3.2 Graphical User Interface

- `client.gui` (optional, for client only). Whether client should display a window or not (yes or no and true or false). Default value: true. Set to true.

3.3.3 Task specification

Parameters described in this section specify the scope of jobs (tasks) that a client can remotely execute.

- `tasks` (mandatory, for server and client). A number of specified task types. No default value. Set to 2. Usually modified to add new plugable module.

- `task1` (mandatory[6], for server and client). Name (label) of the task type 1. No default value. Set to branchnbound.

- `task2` (mandatory[6], for server and client). Name (label) of the task type 2. No default value. Set to rsparallel.

- ...

- `task1.serverclass` (mandatory[6], for server only). Name of the class file responsible for scheduling jobs (and processing their results) for task type 1. This class should be a subclass of `rslib.dix-er.client.ServerTask` (see DIXER Module Developer Guide for details). The class declared here should be provided at the execution time by including in `dixer.jar` or by entering an entry in the class path of JVM[7]. No default value. Set to `rslib.dixer.branchnbound.ServerBranchNBound`.

- `task1.clientclass` (mandatory[6], for client only). Name of the class file responsible for evaluating jobs of task type 1. This class should be a subclass of `rslib.dixer.client.ClientTask` (see DIXER Module Developer Guide for details). In current release of DIXER dynamic class loader is removed. This means that declared here class should be provided at the execution time by including in `dixer.jar` or by entering an entry in the class path of JVM. No default value. Set to `rslib.dixer.branchnbound.ClientBranchNBound`.

[5]See InetAddress.getHostName() for details.
[6]Depends on the number of configured tasks.
[7]Java Virtual Machine (see [4]). Usually one can start JVM by executing `java`.

- `task2.serverclass` (mandatory[6], for server only). Name of the class file responsible for scheduling jobs (and processing their results) for task type 2. This class should be a subclass of `rslib.dixer.client.ServerTask` (see DIXER Module Developer Guide for details). The class declared here should be provided at the execution time by including in `dixer.jar` or by entering an entry in the class path of JVM. No default value. Set to `rslib.dixer.rsparallel.ServerRSPar`.
- `task2.clientclass` (mandatory[6], for client only). Name of the class file responsible for evaluating jobs of task type 2. This class should be a subclass of `rslib.dixer.client.ClientTask` (see DIXER Module Developer Guide for details). In current release of DIXER dynamic class loader is removed. This means that declared here class should be provided at the execution time by including in `dixer.jar` or by entering an entry in the class path of JVM. No default value. Set to `rslib.dixer.rsparallel.ClientRSPar`.
- ...
- `tasks.maxfailed` (mandatory, for server only). Number of maximal allowable fails of job before dropping job permanently. When a job evaluation fails than it is inserted at the end of the current scheduled jobs queue, but up to the specified here number of times. No default value. Set to 10.
- Module specific settings. Each plugable module (declared before as a task type) can require additional settings. The parameters begins with name of the module instead of `task` prefix. Usually the parameters begins with the same name as the name specified in `taskX=<name>` parameter[8]. For example the value `maxfailed` is usually implemented by each module (in form of `<name>.maxfailed` and can overwrite the standard value specified in `tasks.maxfailed`. See a particular module User Manual for details.

3.3.4 Command specification

Parameters described in this section specify binary codes of commands that server and client send to each other. It is very important that client and server work with identical codes.

- `command.echo` (mandatory, for server and client). Code for command requesting an identification string (with version number) from a client. No default value. Set to 1.
- `command.kill` (mandatory, for server and client). Code for command requesting client to die. No default value. Set to 2.
- `command.verifyfile` (mandatory, for server and client). Code for command requesting verification of particular file. No default value. Set to 3. Verification consist in checking file name, existence, its length and in comparing CRC16 of the first 4KB of file.
- `command.transferfile` (mandatory, for server and client). Code for command requesting storing of a file relative (or not, if it is directly requested) to current directory. No default value. Set to 4.
- `command.servermessage` (mandatory, for server and client). Code for command displaying message on a client window. No default value. Set to 5.
- `command.answer` (mandatory, for server and client). Stamp code of answer for command. No default value. Set to 65536.
- `command.<task 1 name>` (mandatory[9], for server and client). Code for command related with task 1. If task 1 is specified as `branchnbound` than this parameter should be written as `command.branchnbound`. No default value. Set to 100.

[8]The prefix name for the task specific parameters is encoded in source code, while the name for a task type can be arbitrarily chosen in the configuration file. It is recommended to use the same name for both purposes to maintain the clarity of the configuration file.

[9]Depends on the configured task type name.

- `command.<task 2 name>` (mandatory[9], for server and client). Code for command related with task 2. If task 2 is specified as `rsparallel` than this parameter should be written as `command.rsparallel`. No default value. Set to 101.
- ...

3.3.5 Debugging and verbosity

- `config.warnings` (optional, for server and client). Whether warnings should be displayed on a java console (yes or no). Default value no. Set to yes.
- `config.debug` (optional, for server and client). Whether debug messages should be displayed on a java console (yes or no). Default value no. Set to yes.

Chapter 4

Operating Server

The DIXER Server should by running only once, at the administrator's computer. It manages all clients that are reachable in used network. It is possible to safely run more than one server if they operate on different computers or on different port numbers (see 3.3). In such a case a server will manage only those clients that have it host address selected as a server address and operate on the same port number.

The DIXER Server can be executed by running a java class named `rslib.dixer.client.Server`. For convenience this is usually done in a batch script.

4.1 Command line parameters

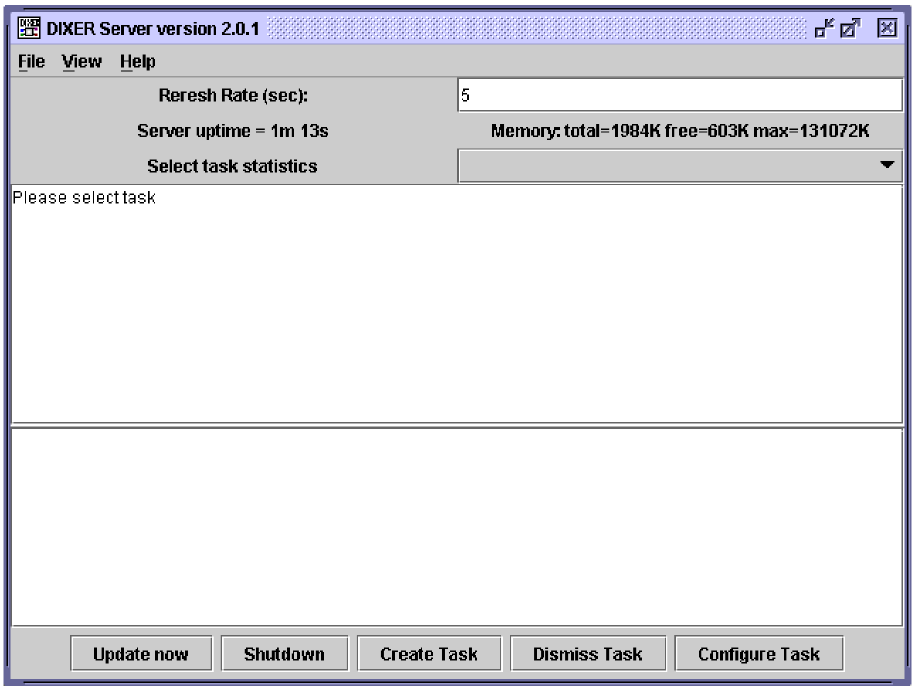
It is possible to pass command lines parameters to the main server class `rslib.dixer.client.Server` (see tools description in [6] for details). The server parameters are not case sensitive except the file names, which depends on the used operating system.

- `-VERSION` — shows version and exits. The version number is read from the configuration file (c.f. 3.3).
- `-C <filename>` — specifies the name for the configuration file. At first the server will search for such a file in class path (c.f. ClassLoader.getResourceAsStream() in Java API Documentation [6]). Than it will search in a current (or else, if specified in name) path for a such file in a regular file system. If this parameter is not specified it is assumed to use `dixer.conf`configuration file. A standard version of this file is included inside `dixer.jar`.
- `-DEBUG` — enables debugging messages (overrides the settings in configuration file).
- `-NODEBUG` — disables debugging messages (overrides the settings in configuration file).
- `-WARNINGS` — enables warnings messages (overrides the settings in configuration file).
- `-NOWARNINGS` — disables warnings messages (overrides the settings in configuration file).
- `-VERBOSE` — enables warnings and debugging messages (overrides the settings in configuration file).
- `-SILENT` — disables warnings and debugging messages (overrides the settings in configuration file).
- `-QUIET` — a synonym for `-SILENT`.

4.2 Server window

At the figure 4.1 one can see a server window. If server is properly configured such a window appears in few seconds after execution of the server class.

The server window provides following information:



Rereshh Rate (sec): 5
Server uptime = 1m 13s
Memory: total=1984K free=603K max=131072K
Select task statistics
Please select task

Figure 4.1: DIXER Client without connection

- Refresh rate — time in seconds in which the whole window will be periodically refreshed. User can change this time by entering a number in right text input filed and pressing enter key.
- Server uptime — says how much time server is operating.
- Memory: total x free y max z — size of total memory allocated by JVM (x in KB), size of free memory in already allocated memory (y in KB, 0<=y<=x) and maximal size of memory that JVM can allocate (z in KB, 0<=x<=z). Helpful to diagnose status of consumed resources.
- Task statistics selection — on the left side is a request for selecting appropriate task and on the right side user can select task for which it would like to see statistics messages.
- Task specific statistics — the content of this text area is directly implemented in a selected plugable module. User should carefully read the information displayed here, because this reflect the current status of computations for selected task[1].
- Report on connected clients — the DIXER Clients currently connected to the server are displayed identified by its dictionary name, network name or IP address. Also some additional data is provided like number of processed jobs, approximated time spent on computations and last communication time.
- Update now button — pressing this button will update the content of window immediately.
- Shutdown button — pressing this button will kill the server.
- Create task button — pressing this button will open a dialog where user can choose a task type for newly created task.

[1]It should be stress here one more time that there can be more than one task related with the same task type — plugable module. All of them operate separately and display different statistics messages.

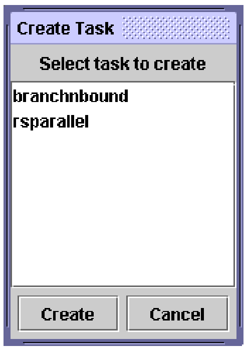
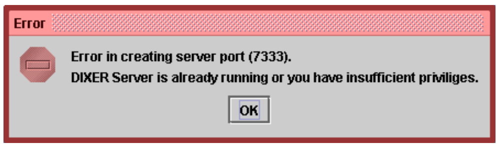


Figure 4.2: Selecting a task to creation.



Error in creating server port (7333).
DIXER Server is already running or you have insufficient priviliges.

Figure 4.3: The error message is displayed when the user executes the DIXER Server more than once.

- Dismiss task button — pressing this button will stop currently selected task. The task is not killed immediately. It waits for all clients to finish their jobs related with this task (if any). Task is permanently destroyed if there are no more jobs related with this task (see also `server.clienttimeout` parameter in the configuration file).

- Configure task button — pressing this button will open the task specific window related with currently selected task. If this option is not implemented in a plugable module than the standard window appears "Task configuration not available".

4.3 Creating new tasks

By pressing "Create task" it is possible to create new tasks, i.e. a new instance of a task type. The newly created task will be executed on the server and than it will schedule jobs for connected clients.

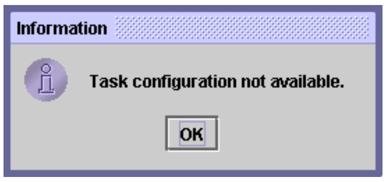
After pressing button "Create task" the task selection dialog appears as on figure 4.2. When the selection is approved by pressing button "Create" on this dialog (fig. 4.2) the task is created. For some task types it may be followed by a task-specific configuration dialog (as it is for e.g. Branch'n'Bound and RSParallel modules). After successful creation of a new task the first available task number will be assigned to this task and the task will appear in task selector in main server window (c.f. 4.2).

4.4 Monitoring server status

4.4.1 Server status

It the server works fine it shows the time how much it is working (up-time) and current memory usage at the bottom of the window (c.f. 4.2). These fields are systematically updated each "refresh rate" seconds.

It is not allowed to run the DIXER Server more than once on the same network port. Such a situation is detected and the error message as on the figure 4.3 is displayed. It is however possible to run the DIXER Server more than once on the different port numbers. It may also happened that the user does not have necessary privileges to execute the DIXER Server on the specified port number. See the configuration file details (sec. 3.3) to solve this problem.


Information

Task configuration not available.

Figure 4.4: A default task configuration window is displayed when a module does not provide an additional run-time configuration.

4.4.2 Tasks

One can choose in task selector to see the statistics for a particular task. The tasks are described in task selector by its consecutive number and task type name. After selecting a particular task its statistics appear in the upper text area (c.f. 4.2).

4.4.3 Cluster hosts

List of the connected hosts that form a DIXER Cluster is displayed in the lower text area (c.f. 4.2). These statistics show time of the last contact with a host and a number of computed jobs for each defined task type. They are extremely useful for monitoring problems with network communication or client software and memory assignment[2].

4.5 Dismissing tasks

Pressing "Dismiss task" button will stop currently selected task. The currently selected task is the task displayed at the task statistics selector and in the tasks statistics text area (c.f. 4.2). The task is not killed immediately. It waits for all clients to finish their jobs related with this task. Task is permanently destroyed if there are no more clients that process jobs related with this task (see also `server.clienttimeout` parameter in the configuration file).

4.6 Configuring tasks

Some of modules provides additional configuration possibility at the run-time of a task instance. Pressing "Configure task" button will open the task specific window related with currently selected task. The currently selected task is the task displayed at the task statistics selector and in the tasks statistics text area (c.f. 4.2). If this option is not implemented in a plugable module (that is a case for current version of Branch'n'Bound and RSParallel modules) than the standard window appears "Task configuration not available" (see figure 4.4).

[2]The assignment of the reasonable amount of memory for DIXER Client is a very important issue, see also 8.

Chapter 5

Operating Client

The DIXER Client can be executed by running a java class named `rslib.dixer.client.Client`. This is usually done in a batch script. Some plugable modules have problems with leak of memory[1] so in standard distribution the batch file loops forever in executing client class. If a client dies because of out of memory error than it will be restarted.

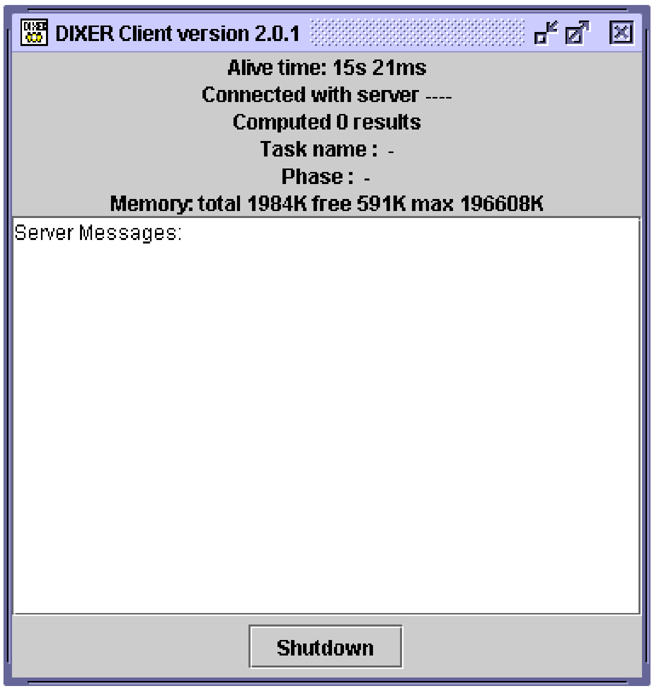
It is strictly forbidden to execute more than one DIXER Client in the same working directory (on the same or different computer). It is also not recommended to execute more than one DIXER Client on the same computer as the some RS-Lib operations may interfere.

5.1 Command line parameters

It is possible to pass command lines parameters to the main client class `rslib.dixer.client.Client` (see tools description in [6] for details). The client parameters are not case sensitive except the file names, which depends on the used operating system.

- `-VERSION` — shows version and exits. The version number is read from the configuration file (c.f. 3.3).
- `-C <filename>` — specifies the name for the configuration file. At first the server will search for such a file in class path (c.f. ClassLoader.getResourceAsStream() in Java API Documentation [6]). Than it will search in a current (or else, if specified in name) path for a such file in a regular file system. If this parameter is not specified it is assumed to use `dixer.conf`configuration file. A standard version of this file is included inside `dixer.jar`.
- `-GUI` — enables Graphical User Interface (overrides the settings in configuration file).
- `-NOGUI` — disables Graphical User Interface (overrides the settings in configuration file).
- `-NO_GUI` — a synonym for `-NOGUI` (left for backwards compatibility).
- `-DEBUG` — enables debugging messages (overrides the settings in configuration file).
- `-NODEBUG` — disables debugging messages (overrides the settings in configuration file).
- `-WARNINGS` — enables warnings messages (overrides the settings in configuration file).
- `-NOWARNINGS` — disables warnings messages (overrides the settings in configuration file).
- `-VERBOSE` — enables warnings and debugging messages (overrides the settings in configuration file).
- `-SILENT` — disables warnings and debugging messages (overrides the settings in configuration file).

[1]Java Garbage Collector works pretty poor with complex data structures.


DIXER Client version 2.0.1

Alive time: 15s 21ms
Connected with server ----
Computed 0 results
Task name : -
Phase : -
Memory: total 1984K free 591K max 196608K

Server Messages:

Figure 5.1: The DIXER Client not connected to the DIXER Server.

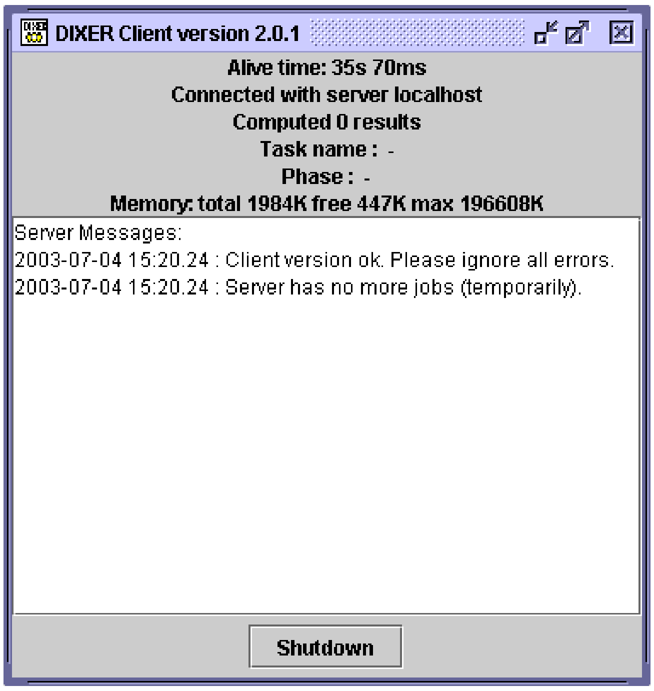
- `-QUIET` — a synonym for `-SILENT`.
- `<serveraddress>` — the first parameter that is not recognized as a one of the above will be used as a server address. If server address is not specified than the content of the `dixer.connection` will be used. See section 3.2.1 for details.

5.2 Client window

At the figure 5.1 we can see a client window. If client is properly configured such a window appears in few seconds after execution of the client class.

The client window provides following information:

- Alive time — time that client staying alive. Helpful in diagnosing out of memory or other errors frequency.
- Connected with server — name of a server when client is connected (cf. fig. 5.2) or "- - -" string when client is not connected (cf. fig. 5.1).
- Computed x results — number of computed results related with all defined tasks.
- Task name — name of currently processed task or empty string if idle.
- Phase name — name of current phase of processed job (if any). Helpful to diagnose status of client computations.
- Memory: total x free y max z — size of total memory allocated by JVM (x in KB), size of free memory in already allocated memory (y in KB, 0<=y<=x) and maximal size of memory that JVM can allocate (z in KB, 0<=x<=z). Helpful to diagnose status of consumed resources.
- Server messages — free text messages from server that inform about intercommunication and server status.



Alive time: 35s 70ms
Connected with server localhost
Computed 0 results
Task name : -
Phase : -
Memory: total 1984K free 447K max 196608K

```
Server Messages:
2003-07-04 15:20.24 : Client version ok. Please ignore all errors.
2003-07-04 15:20.24 : Server has no more jobs (temporarily).
```

Figure 5.2: The DIXER Client connected to the DIXER Server. The Server currently has no jobs for this computer.

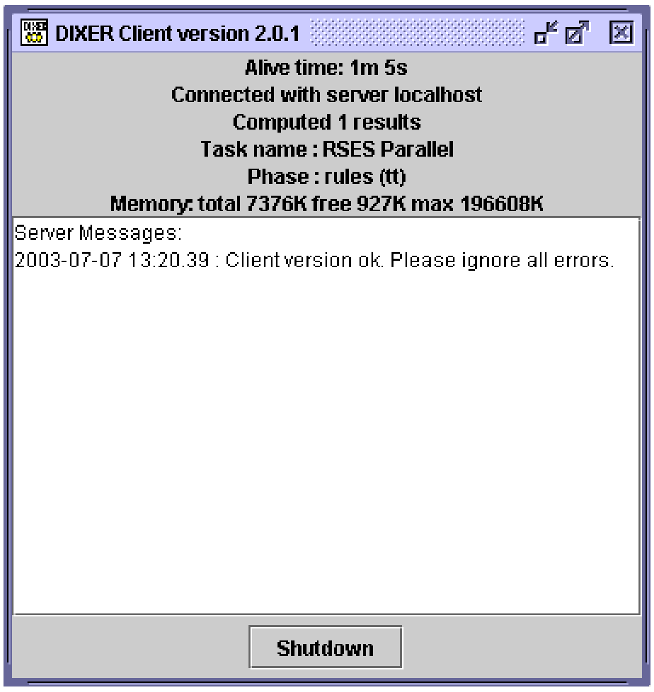
- Shutdown button — pressing this button will kill the client. If the client is executed in an infinite loop (in batch script) than this action restarts client.

5.3 Monitoring client status

At the beginning server checks client version by an echo command. In result of that in a "Server messages" will appear one of the following messages: "Client version ok" — if the client version is in accordance with server version or "Client version obsolete" otherwise.

Mainly we can say that client operates under two conditions: there are jobs to compute or there are no jobs to compute. If there are jobs to compute the current job is described in the client window as described in previous section (cf. fig. 5.3). If there are no more jobs the server will inform client by a free-text message (cf. fig. 5.2). How frequently server informs a client can be changed in the configuration file.

If the communication with server is broken (i.e. due to the server shutdown) the client is be able to notice this fact when it is communicating with the server. During computations the communication is disabled to save the network overload and time-delays. It means that if the communication is broken during computations the client will notice it after finishing its computations (both successfully or erroneously).


DIXER Client version 2.0.1

Alive time: 1m 5s
Connected with server localhost
Computed 1 results
Task name : RSES Parallel
Phase : rules (tt)
Memory: total 7376K free 927K max 196608K

```
Server Messages:
2003-07-07 13:20.39 : Client version ok. Please ignore all errors.
```

Figure 5.3: DIXER Client

Chapter 6

Branch'n'Bound Module

6.1 Introduction

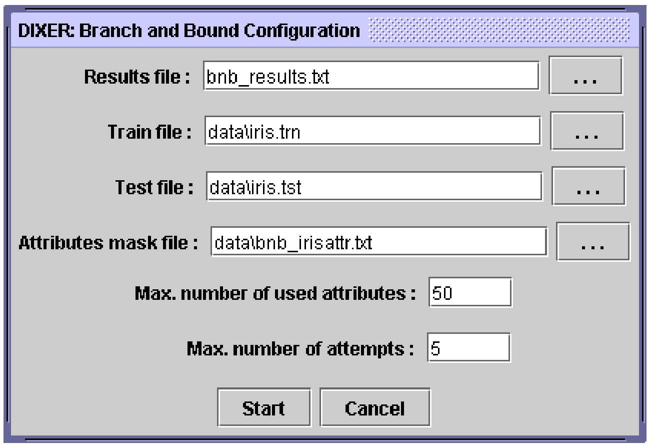
The Branch'n'Bound Dixer plugable module allows to carry out experiments with very expensive feature selection based on the wrapper method (see e.g. [3]). The strategy for searching int the space of the possible attribute subsets is similar to the branch and bound search. Here, however, this strategy is not complete and is used rather to selecting the first candidate to visit in the lattice of the possible feature subsets. The results are saved in the special file that can be vieved both manually and with a help of the text-processing tools, such as: `awk` or `gawk` that is the GNU Project's implementation of the AWK programming language, Microsoft® Excel or OpenOffice Spread spreadsheet editor or any programming language that support textual string manipulation and I/O operations.

As the wrapper classifier the decision tree is used here. The settings of the used decision tree are 0.98 for the minimal purity of the leaf and 5 for the indivisible leaf size. The decision tree with clustering of the symbolic values and the indiscernibility as the split evaluator is used.

The Branch'n'Bound module is implemented package `rslib.dixer.branchnbound`, but uses also classes from other packages to perform experiments. The server class is `rslib.dixer.branchnbo-und.ServerBranchNBound` and the client class is `rslib.dixer.branchnbound.ClientBranchNBound`. These class identifiers should be specified in configuration file `dixer.conf` as `task<N>.serverclass` and `task<N>.clientclass` respectively (see sec. 3.3 for details).

6.2 Module specific parameters

- `branchnbound.resultsfilename` (optional, for server only). Default name for the results file. Useful in serial or batch experiments. No Default value. Set to bnb_results.txt.
- `branchnbound.trainfilename` (optional, for server only). Default name for the file name of training set. Useful in serial or batch experiments. No Default value. Set to data\iris.trn.
- `branchnbound.testfilename` (optional, for server only). Default name for the file name of testing set. Useful in serial or batch experiments. No Default value. Set to data\iris.tst.
- `branchnbound.attributefilename` (optional, for server only). Default name for the file name of attribute list. Useful in serial or batch experiments. No Default value. data\bnb_irisattr.txt.
- `branchnbound.attributelimit` (optional, for server only). No Default value. Set to 50.
- `branchnbound.maxfailed` (optional, for server only). Number of maximal allowable fails of job before dropping job permanently. When a job evaluation fails than it is inserted at the end of the current scheduled jobs queue, but up to the specified here number of times. Default value `tasks.maxfailed`. Set to 5.


DIXER: Branch and Bound Configuration

Results file : bnb_results.txt

Train file : data\iris.trn

Test file : data\iris.tst

Attributes mask file : data\bnb_irisattr.txt

Max. number of used attributes : 50

Max. number of attempts : 5

Figure 6.1: Creating the Branch'n'Bound task in DIXER Server.

6.3 Task creation

The create dialog for this task is presented on the figure 6.1. In configuration of this type of task is necessary to provide following settings:

- Results file — the file where the results (evaluated elements of the subset lattice) are stored,
- Train file — the decision table used for training the decision tree,
- Test file — the decision table used for evaluation of the induced decision tree (can be the same as the training file),
- Attributes mask file — the plain list of attributes (each attribute in a separate line) that should be used in searched lattice of subsets.
- Maximal number of attributes — the maximal number of used attributes in a evaluated subset (i.e. an upper cut of the lattice),
- Maximal number of attempts — the same as the `maxfailed` parameter.

6.4 Results file

The results file contains the already computed nodes of the subset lattice. A node can be not opened, opened or closed. The not opened node it is a node that was not evaluated. An opened node is a node that system has already computed the classifier accuracy for corresponding subset of attributes. A closed node is a node for which all successors (subsets that have at most on one attribute more) are either opened or closed.

In the results file are stored opened and closed nodes in the following form:

- `o=0,0,1,1,1:0.9` — an opened node,
- `c=0,0,1,1,1:0.9` — a closed node.

The letter at the beginning of the line indicate whether a node is opened ("o") or closed ("c"). After the equation mark the node description is written as the characteristic function of used attributes. The 1 means that subsequent attribute was used in classifier induction, while 0 means that respective attribute wasn't used. At the end of the line, after the semicolon character, the accuracy of the classifier is written. The accuracy is measured as the number of answers compatible with the test decision table.

Chapter 7

RSParallel Module

7.1 Introduction

The RSParallel Dixer plugable module allows to carry out experiments based on methods implemented in Rough Set Exploration System (RSES) and particularly in RSES-Lib (c.f. [2]). It process a special script language described in section 7.3 that allows to prepare a individual scenario for experiments. It is strongly recommended to prepare scripts in some semi-automatic manner. There are a lot of suitable tools for this task such as: `awk` or `gawk` that is the GNU Project's implementation of the AWK programming language, Microsoft® Excel or OpenOffice Spread spreadsheet editor or any programming language that support textual string manipulation and I/O operations.

The RSParallel module is implemented package `rslib.dixer.rsparallel`, but uses also classes from other packages to perform experiments. The server class is `rslib.dixer.rsparallel.ServerRSPar` and the client class is `rslib.dixer.rsparallel.ClientRSPar`. These class identifiers should be specified in configuration file `dixer.conf` as `task<N>.serverclass` and `task<N>.clientclass` respectively (see sec. 3.3 for details).

7.2 Module specific parameters

The RSParallel Module uses following parameters in configuration file (c.f. sec. 3.3):

- `rsparallel.forcerecompute` (optional, for server only). Whether the job that was previously computed should be recomputed from the beginning or not (yes or no and true or false). Default value false. Set to false.

- `rsparallel.maxfailed` (optional, for server only). Number of maximal allowable fails of job before dropping job permanently. When a job evaluation fails than it is inserted at the end of the current scheduled jobs queue, but up to the specified here number of times. Default value `tasks.maxfailed`. Set to 20.

- `rsparallel.scriptfilename` (optional, for server only). Default name for the script file. Useful in serial or batch experiments. No default value. Set to rsp_rules.txt.

- `rsparallel.resultsfilename` (optional, for server only). Default name for the results file. Useful in serial or batch experiments. No default value. Set to rsp_rules.log.

7.3 Script language description

The RSParallel Module uses a special script language to schedule experiments across cluster of computers. This script language is interpreted line by line from the beginning of the file to its end and each line contains description of separate experiment.

The line that describes an experiment consist of three main parts:
<Experiment Type> <Data Tables> <Experiment Parameters>

The meaning of these three parts will be explained in following sections. Besides the experiment description lines the script language allows special lines that are insignificant from the experimental point of view:

- *Comment.* The line can be a comment line if its begins with character # or if it is an empty line.
- *Echo.* If the line contains a string "echo" at the beginning of this line the whole content of this line will be copied to output log file. This allows to add some comments in this log file.

7.3.1 Experiment Type

There are two experiment types in this version of script language:

- `TT` — Train&Test mode. The experiment will be executed in Train&Test mode. This means that user should support a separate data set (table) for training classifier and separate data set (table) for testing classifier.
- `CV` — Cross Validation mode. *This mode is not supported in version 2.0.* The experiment will be executed in Cross Validation mode. The Cross Validation uses only one data set for training and testing. This set is partitioned into a number (let say N) of subsets of possibly equal size and in N steps each of this subsets is assumed to be a testing set. As a training set $N-1$ remaining subsets are taken in such a way that the same subset cannot be used for training and testing in the same step. After all N steps each part of original data set is used for training and testing.

This keywords are not case-sensitive.

7.3.2 Data Tables

In the second part of experiment description the description of data sets (tables) are provided. The specification of data sets depends on the mode in which the experiment will be carried out.

- If as the experiment mode the Train&Test was chosen than it is necessary to specify a file name for the training table and for the testing table: `my_training.tab my_testing.tab`. The data tables should be written in RSES Data Table Format described in [1].
- If as the experiment mode the Cross Validation was chosen than it is necessary to specify a file name of one table used for both training and testing and the number of steps (folds). For example "`my_data.tab 5`" means that the 5-fold Cross Validation will be performed on the "my_data.tab" data file.

The host Operating System rules apply whether the file names are case-sensitive or not. But to achieve the portability we should assume that file names are case-sensitive.

7.3.3 Experiment Parameters

Description of the experiment parameters is the most important part, as it describes the classifier to be used. This description does not depends on the experiment mode. In both Train&Test and Cross Validation experiments any classifier can be used. The keywords are not case-sensitive.

Rule-based classifiers

`RULES` *<algorithm> <discretization> <shortening factor>*

- *Algorithm.* There are three different algorithms:
  1. `ALL` — induces all rules.

2. `COV` — induces minimal covering set of rules.
3. `GEN` — induces genetically optimized set of rules.

- *Discretization.* There are four different discretization (scaling) methods:
  1. `NO_SCAL` — without discretization at all (tables should contain only symbolic attributes).
  2. `GLOBAL_SCAL` — global discretization (in current version of RSES-Lib available only on MS-Windows^{TM} platform).
  3. `LOCAL_SCAL` — optimized local discretization.
  4. `LOCAL_SCAL_WS` — optimized local discretization with additional discretization (clustering) of symbolic attributes.
- *Shortening factor.* A number between 0.0 and 1.0 that adjust the shortening of induced rules. The 1.0 means no shortening and 0.0 means empty rules.

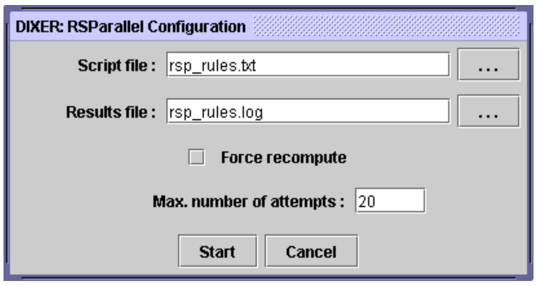
Decomposed-rules classifier

The decision tables can be decomposed by a template based decomposition tree before inducing the decision rules. It is advised especially for the huge decision tables for which the standard rule induction works unacceptably long.
`DECOMP_RULES` *<leaf size> <discretization> <shortening factor>*.

- *Leaf size.* The maximal leaf size. If a node contains equal or less objects (cases) than it will automatically become a leaf and will be used as a decision table for inducing the minimal covering set of rules (as above `RULES COV`).
- *Discretization.* Set to true if the decision subtables should be discretized after the decomposition.
- *Shortening factor.* The shortening factor of the decision rules.

7.3.4 Examples

- **`tt train.tab test.tab rules all local_scal 1.0`**
  The Train&Test experiment with training table “train.tab” and testing table “test.tab” that use the all decision rules induction algorithm as a classifier. The data will be discretized locally (with optimization) and rules will not be shortened.
- **`tt train.tab test.tab rules all global_scal 0.8`**
  The Train&Test experiment with training table “train.tab” and testing table “test.tab” that use the all decision rules induction algorithm as a classifier. The data will be discretized globally and rules will be shortened by a factor 0.8.
- **`tt train.tab test.tab rules cov local_scal 0.7`**
  The Train&Test experiment with training table “train.tab” and testing table “test.tab” that use the minimal covering decision rules induction algorithm as a classifier. The data will be discretized locally (with optimization) and rules will be shortened by a factor 0.7.
- **`tt train2.tab test2.tab rules gen global_scal 0.9`**
  The Train&Test experiment with training table “train2.tab” and testing table “test2.tab” that use the genetic decision rules induction algorithm as a classifier. The data will be discretized globally and rules will be shortened by a factor 0.9.
- **`tt train2.tab test2.tab rules all local_scal_ws 0.6`**
  The Train&Test experiment with training table “train2.tab” and testing table “test2.tab” that use the all decision rules induction algorithm as a classifier. The data will be discretized locally with clustering of all symbolic attributes and rules will be shortened by a factor 0.6.


DIXER: RSParallel Configuration

Script file : rsp_rules.txt

Results file : rsp_rules.log

Force recompute

Max. number of attempts : 20

Figure 7.1: Creating the RSParallel task in DIXER Server.

- `tt train3.tab test3.tab rules all no_scal 0.5`
  The Train&Test experiment with training table "train3.tab" and testing table "test3.tab" that use the all decision rules induction algorithm as a classifier. The data will be not be discretized and rules will be shortened by a factor 0.5.

- `tt my_tab.rses my_tab.rses decomp_rules 100 true 1.0`
  The Train&Test experiment with decomposition tree. The "my_tab.rses" decision table will be decomposed into the subtables of no more than 100 objects. The formed subtables will be used to induce minimal covering decision rules with discretization and shortening factor 1.0.

7.4 Task creation

The create dialog for this task is presented on the figure 7.1. The necessary settings are the name of the script file to execute (as describe in the previous section) and the result file (as described in the next section). Additionally user can specify if the results that are already present in the result file should be recomputed.

7.5 Result file

The results of the script processing are written to the result file. This file is designed to both looking manually or processing automatically. The corresponding lines in the script file generates output lines (one or more) in the result file.

The echo lines in script file write message included in echo command. The content of the echoed message is not interpreted and could be used to special tagging or making the result file more readable.

The experiment lines in script file write the experiment description and experiment results. The structure of such line is as follow:

<*Result flag*><*Experiment description*> | <*Experiment results*>

- *Result flag.* If the computations were successful the '+' character is written here. If the computations failed the '-' sign is written here.

- *Experiment description.* Exactly the same experiment description as in the script file.

- *Experiment results.* After the separator character come the computed results. If the computation were successful the results include the classifier accuracy, number of rules, true positive rates and other parameters that describe the classifier performance. If the computations failed the error message is written here.

Chapter 8

FAQ

- Q: The DIXER Server as well as DIXER Client does not even start.
- A: Make sure that there is a Java Runtime Environment in version at least 1.4 installed on this computer. Make sure that installation is not corrupted and `java` executable is in the executable path. If it is necessary modify the `PATH` environment variable.
- Q: The DIXER Client shows the following message: "DIXER Cluster failure. Incompatibile DIXER Server/Client or Java Virtual Machine version."
- A: Make sure that you are using Java version 1.4 or higher. This DIXER Client is taken from the other, incompatible distribution of the DIXER Software. Reinstall the DIXER Client in the release version compatible with the DIXER Server.
- Q: The DIXER Client cannot compute even one job.
- A1: Try to increase maximal available memory for JVM by executing with `-Xmx` parameter[1]. For example `java -Xmx256M rslib.dixer.client.Client`.
- A2: The DIXER Client cannot find the dynamic (shared) library RSLib. Please verify location of the file `RSLib.dll` or `RSLib.so` and adjust the `-Djava.library.path` JVM parameter.

[1] See `java -X` for details. The -X options are non-standard and subject to change without notice.

Bibliography

[1] J. G. Bazan, M. Mikołajczyk, and M. Szczuka. *Rough Set Exploration System version 2.0, User Manual.* Warsaw University, 2003.

[2] J. G. Bazan, M. S. Szczuka, and J. Wróblewski. A new version of rough set exploration system. In J. J. Alpigini, J. F. Peters, A. Skowron, and N. Zhong, editors, *Rough Sets and Current Trends in Computing, Third International Conference, RSCTC 2002*, *LNAI* 2475, pages 397–404. Springer, 2002.

[3] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131–156, 1997.

[4] T. Lindholm and F. Yellin. *The Java^{TM} Virtual Machine Specification, Second Edition.* Addison-Wesley, 1999.

[5] Sun Microsystems Inc. *Java^{TM} 2 SDK, Standard Edition Documentation, Version 1.4.0*, February 13, 2002.

[6] Sun Microsystems Inc. *Java^{TM} 2 SDK, Standard Edition Documentation, Version 1.4.1*, September 16, 2002.