The Seventh International Planning Competition Documentation

Release 1.3

Carlos Linares Lopez

July 16, 2012

CONTENTS

CHAPTER

ONE

INTRO

The International Planning Competition is an important event for the Planning and Scheduling Community. As such, it receives a lot of attention and, as a matter of fact, it has received an outstanding number of submissions in the Seventh edition.

While the Competition is intended, in the very end, to declare both a winner and a runner-up per track, its very important role is to gather data and disseminate it to all researchers. Besides, it has been also used to promote or to review standards and to foster (sometimes vivid) discussions. At last, but not least, the problems and domains used at the various editions of the International Planning Competition have been used as a reference in most experiments by the vast majority of practicioners and the practice of choice to show progress with respect to a particular planner is to show the performance for the same set of problems of the last Competition.

Therefore, while it is mandatory to provide the names of the research team and the code of the top performers at the different tracks, there are various advantages in distributing also the code used to evaluate them. Precisely, the target of this documentation pages is to serve as a reference point for using and extending the code used at the **Seventh International Planning Competition**.

Thanks to a common repository which shares the code for researchers to make experiments and for the organizers of any International Planning Competition to evaluate every planner:

- **All competitors can test their software before submitting it to the International Planning Competition** —or *IPC*.

  This would significantly improve the overall running time of an International Planning Competition. Common errors are scripts without a shebang line, executables without the right permissions set, compilation errors (either due to the assumption that the target machine have a number of libraries available or just by mistaking whether the target architectures are 32/64 bits, etc.) and a long etcetera. However, if the organizers of an IPC set up a SVN repository and allow competitors to enter their computers before the submission deadline, they can actually test their software just by executing one script. *The script produces a log file which can then be used as a ticket for registering.*

- **Everyone can easily run tests on his planner to make comparisons with other planners**

  This sets up a common framework for everyone to make experiments making comparisons fairer. Of course, planners under development are expected to do not be exposed to the general public. Therefore, one of the main drivers of the software of the Seventh International Planning Competition was the ability to work with respect to any SVN. This way, *all research teams can create their own private repostiories so that private experiments can be conducted*. Then, if desired, the results can be made available to the public by importing them into a svn repository.

- **The results of all the experiments are permanently stored in a repository publicly available**

  By providing additional tools to allow everyone to examine the available data there is no need to be repeating experiments over and over. Instead, they are performed only once and then permanently stored at the repository. Of course, the same principle applies to private repositories created by anyone. This also means that the results

of many of these scripts can be deleted from the local computer once the experimentation is over since the planners/domains and results are stored in the SVN.

In particular, this principle is essential to the organizers of the International Planning Competition. Allowing everyone to inspect the SVN repository, the evaluation becomes more transparent. As a matter of fact, *the contents of the working SVN repository used during the Seventh International Planning Competition was dumped to a new location preserving all revisions*.

The idea of creating software for automating the tasks necessary for evaluating planners (from automatically compiling planners, creating testsets, validating results and even generating pdf documents with an overall view of the performance of each entry) has been already considered in other editions of the IPC.

In particular, the organizers of the Sixth International Planning Competition developed a lot of code for automating all these tasks. The architecture devised at that time consisted of three facilities:

1. A SVN repository which stores all the necessary information. Not only domains, planners, and scripts necessary for running the competition, but also other folders which store and share private data of the organizers

2. A wiki which is used to publish and share information among the organizers and the community. Thanks to the *Access Control Lists* the wiki allows different levels for sharing information among users.

3. A cluster or, in general, the hardware premises where the competition is actually run. This can be either a single computer or a cluster or a network of clusters.

The software of the Seventh International Planning Competition is strongly influenced from all the ideas developed at the precedent edition of the IPC. In fact, the first chapter is devoted to a discussion of these parts and how they relate to each other.

CHAPTER

TWO

GENERAL OVERVIEW

As discussed in the previous chapter, one of the suggestions made at the Sixth edition of the IPC was to set up three different components for organizing an IPC: a SVN repository, a wiki site and the computer facilities that actually run all the experiments.

The following sections discuss all these parts separately and how they have been used in the Seventh International Planning Competition. At least, the first section (*SVN repository*) is important for organizers of other Competitions as well as for people administering the IPC software in a closed environment. Users of the IPC software can safely skip all these sections and go directly to *IPCData*. The last section is devoted to this documentation.

Warning: A word of warning for readers of this manual. While all the features described herein refer to the latest version of the software distributed through the official svn server, some might be available only from a particular version. Therefore, if some are not apparently available in your installation invoke your script with the directive `--version` to make sure that it matches the minimum requirements specified in this manual and consider updating if that is not the case.

2.1 SVN repository

This section justifies the suitability of using SVN repositories and then provides details about the arrangement of contents in the two repositories available after the Seventh International Planning Competition.

2.1.1 Advantages of using SVN repositories

The idea of using a SVN repository allows:

- **Everyone to download and use the same data**

  This means that, at least, planners and domains shall be stored at the SVN repository so that everyone can download them.

  Besides, storing the scripts used for running experiments and reporting results would be highly desirable as well since this would allow everyone to use the same framework for making experiments.

- **Provides a single point that can be copied across different computers**.

  This means that the organizers of a particular IPC can easily make backup copies while everyone is welcome to make copies of the repository to their own computers for making her own experiments in a private environment prior to new publications or the development of new planners for the incoming IPCs.

- **A SVN repository with the results of all the experiments guarantees the repetibility of the same experiments and provides means for making cross validations**

  While providing logs of the executions is highly desirable for showing the competitors/researchers that apparently nothing went wrong, allowing everyone to have a look at the particular results enables various techniques for cross validating the results produced at the International Planning Competition or the experiments conducted at the lab.

The contents and arrangement of data in both repositories is discussed in the next subsection. Throughout all this documentation, the tracks and subtracks are abbreviated as follows:

Acronym	legend
seq	sequential track
tempo	temporal track
sat	satisficing subtrack
mco	satisficing multi-core
opt	optimal subtrack

so that, for example, `seq-opt` stands for the sequential satisficing track and `tempo-sat` stands for the temporal satisficing track.

2.1.2 Contents of the data repository

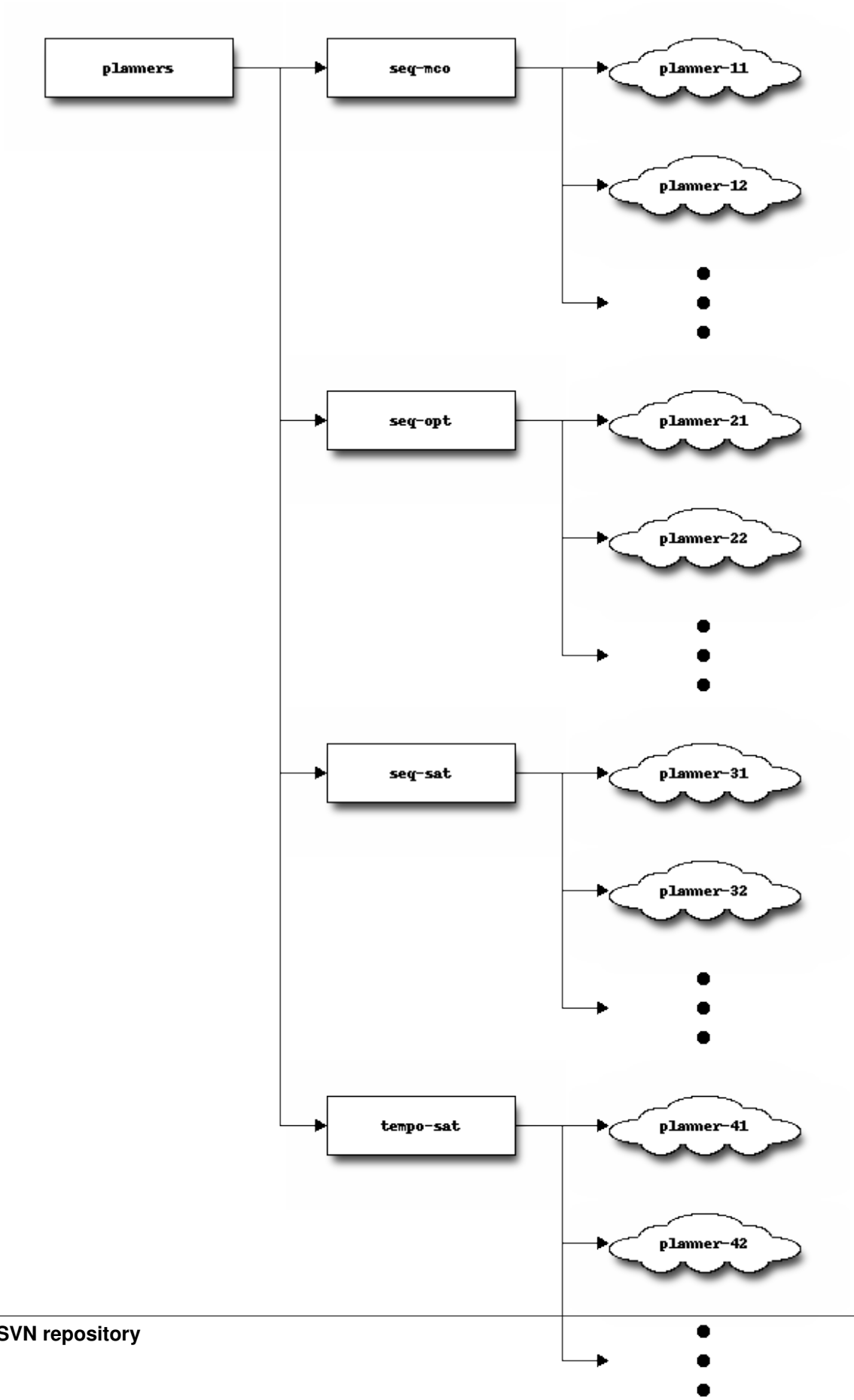
The data repository contains both the planners that took part in the Seventh International Planning Competition, all the domains that were chosen to evaluate them and summaries of all the results gathered so far. Domains, planners and results are organized in the following tracks-subtracks: sequential satisficing (`seq-sat`), sequential optimal (`seq-opt`), sequential satisficing multi-core (`seq-mco`) and temporal satisficing (`tempo-sat`). Finally, the data repository also contains the scripts used at the IPC 2011 and the latest version of this documentation.

A copy of the repository used at the Seventh International Planning Competition is available at:

```
svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
```

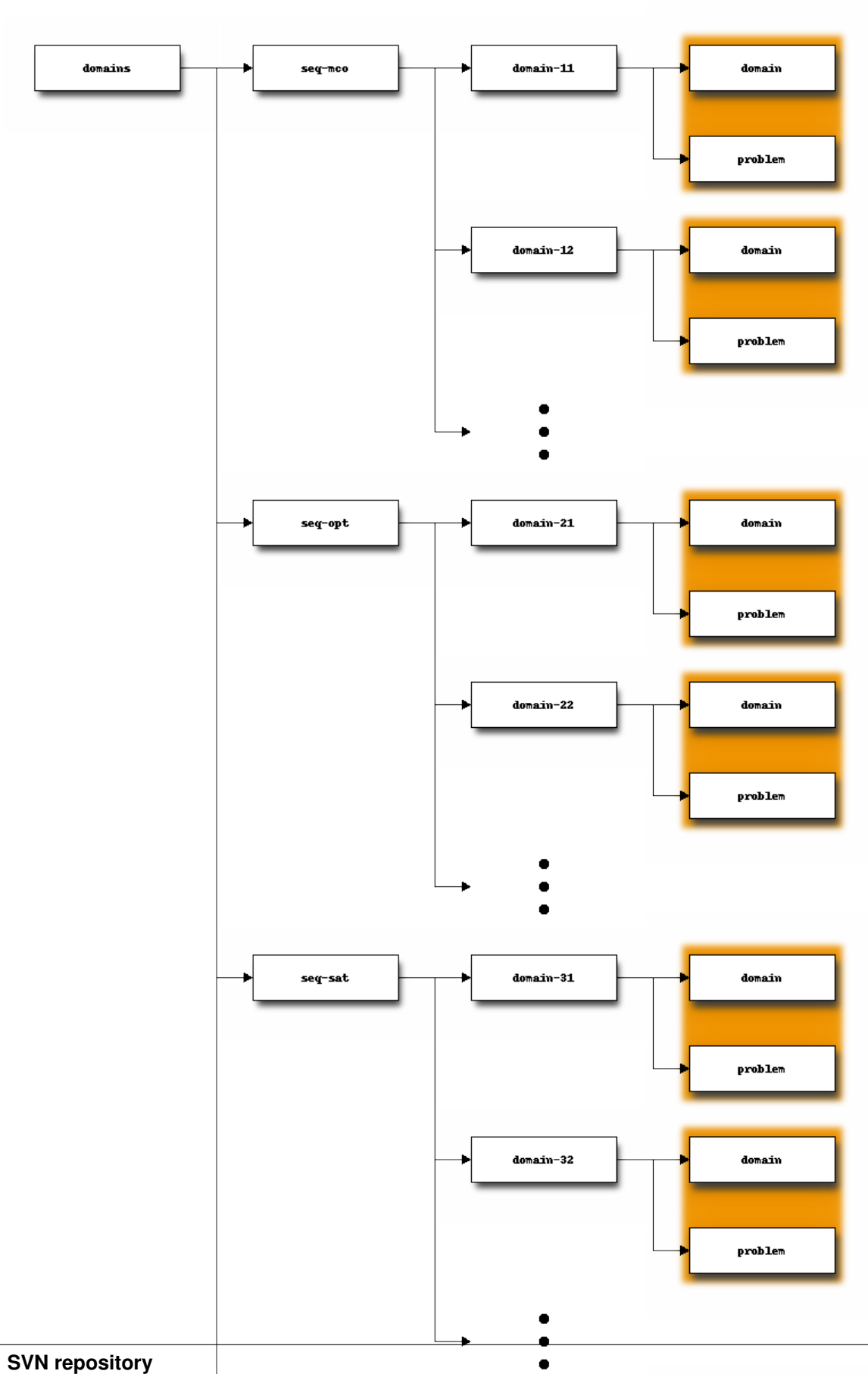
The current size of this repository is almost 1 Gb. Therefore, make sure to allocate space (and time) enough to check-out though it is not necessary at all to retrieve all the domains and planners to make experiments. This section discusses below the organization of planners, domains and scripts. For an introduction to the summaries of the results see *Inspecting data*. A detailed copy of all the results can be accessed through a different svn repository, see *Contents of the results repository*.

The data repository is organized as follows. First, planners are arranged according to the following schema:


planners
seq-mco
planner-11
planner-12
seq-opt
planner-21
planner-22
seq-sat
planner-31
planner-32
tempo-sat
planner-41
planner-42

where *planner-ij* stands for a particular planner preceded by the track-subtrack. For example, the optimal planner **forkinit** is located at `planners/seq-opt/seq-opt-forkinit/`. The clouds shown in the figure above indicate that their content depend on the particular arrangement of every planner. However, the software of the Seventh International Planning Competition requires two particular scripts to be located at the root of these directories: `build` and `plan` —see *check.py*.

Next, domains are stored as follows:


domains
seq-mco
domain-11
domain
problem
domain-12
domain
problem
seq-opt
domain-21
domain
problem
domain-22
domain
problem
seq-sat
domain-31
domain
problem
domain-32
domain
problem
tempo-sat
domain-41
domain

where *domain-ij* shall be substituted by the name of particular domain. For example, the domain floortile used in the sequential optimization track is located at `domains/seq-opt/floortile/`. The directories `domain` and `problems` are grouped to indicate that they define together a particular planning task.

Both, the names of planners and domains shall meet the following requirements:

planners They can consist of any combinations of the alphabetic characters `a-z` either in lowercase or uppercase. Besides, the name also admits any digits and the special characters _ and -. Alternatively, a single dot `.` can be used followed by either an alphabetic character (either in lowercase or uppercase) or a digit.

For example, `lama-2011`, `dae_yahsp` and `Multiplan.s` are legal, whereas `planner#1` or `my.planner` are not allowed. Besides, the blank space is strictly forbbiden.

On the other hand, as a convention (i.e., they are not affected in any way by any script), tracks-subtracks starting with the letter `x` are considered to be *extra* planners, i.e., running in the competition in non-competitive mode. This is the case, for example, of a second version of the planner **ArvandHerd** that was not accepted to take part competitively in the `seq-mco` competition. However, the experiments were run much the same and data with its results were disseminated in the same fashion. They can be found at `planners/xseq-mco/xseq-mco-arvandherd/`

domains They can be made up only of a combination of alphabetic characters (either in lower or uppercase) and digits.

For example, `nomystery` or `barman` are legal names whereas `visit-all` would cause an error in some scripts.

Besides, a track-subtrack (i.e., directories immediately beneath `domains`) that starts with a `t` are considered *test domains* and they are treated as such by the script `invokeplanner.py` —see *invokeplanner.py*. For example, the directory `domains/ttempo-sat/matchcellar/` contains the problems used for running preliminary tests.

This does not mean, however, that the tree structure presented above is rigid. Other directories can be included at any location. In this case, it is very important to make the following observations:

1. Directory names starting with any of the special characters _ or `.` (as in the case of the `.svn` directories) are always skipped at any level.

2. In the particular case of the domains structure, the arrangement discussed above is considered only when looking for particular `domain.pddl` and `problem.pddl` files. In the process, any other directories than those shown above are just ignored.

It is worth noting that tracks and subtracks can be given any name. This serves to differentiate different groups since the IPC scripts **apply only to a particular combination of track-subtrack**. This is, no cross-reference is done between different tracks-subtracks. However, there are two names for subtracks with a distinguished meaning:

mco By default, the time bound given to `invokeplanner.py` is used to set a limit on the available time for all threads and/or processes started by a planner. For example, if the time limit is 30 minutes (1800 seconds) and a planner launches three threads, the sum of all the running times is checked and the planner is killed when this sum is equal or larger than 1800 seconds. However, when specifying the subtrack `mco`, the planner is given the specified time in spite of the number of processes and threads it launches.

dck This term stands for *Domain Control Knowledge* and it is one of the subtracks used at the Learning Competition of the Seventh International Planning Competition. In this particular case, planners shall be given a fourth parameter which specifies the location of the

domain control knowledge, in contraposition to the typical case where only three parameters have to be specified (for more details see *check.py*).

Regarding the scripts, they are all located beneath `scripts/`. One tiny script was developed in awk for processing the results of the Sixth International Planning Competition for choosing problems for the IPC 2011 when reusing domains but has no other applicability and therefore its discussion is skipped here. However, the scripts beneath `scripts/pycentral` are of capital importance and it is expected that they will be also useful for the organizers of other Competitions as well as for researchers at their home labs. For a thorough discussion of the package `IPCData` see *IPCData*; the package `IPCReport` is discussed in *IPCReport*; finally, the package `IPCPrivate` is introduced in *IPCPrivate*.

2.1.3 Contents of the results repository

All the results generated during the IPC 2011 are available at:

```
svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/results
```

Keep in mind that this repository is huge and contains 35+ Gb of data. Therefore, it is not recommend to check out its contents. Instead, it is better suited just to browse its contents for looking for particular results —to this end, there are various free software packages that make this task very easy such as eSVN for Linux and svnX for Mac OS. The organization of the results repository is much the same than any results directory. For a thorough discussion of its contents see *The results directory*.

To access the results in a manageable size, a number of *snapshots* have been generated and are accessible through the first repository either: just to inspect data (see section *Inspecting data*); to compute the score of a set of planners (see sections *score.py* and *tscore.py*) and even to perform statistical tests —see *test.py*.

2.2 Wiki site

Wikis provide effective means for sharing data rapidly in various formats at different levels. Besides, they do provide a friendly environment so that they have been used for formally presenting the contents and progress of the Seventh International Planning Competition.

The wiki for the IPC 2011 can be found at http://www.plg.inf.uc3m.es/ipc2011-deterministic

Any wiki engine would make the service. In particular, at the Sixth edition, the MoinMoin Wiki engine was selected and it was used again at the Seventh International Planning Competition.

One of the main advantages of using a Wiki is that they allow a number of private pages to be written for sharing data among organizers thanks to the so-called acl (*access control lists*) that allows different people to share data among them without ever letting others notice. Examples are: todo pages, domain candidates, partial results, etc.

2.3 Computer facilities

In the very end, an arbitrary number of planners shall be run on a particular selection of domains and problems on a computer. While most practitioners might found it useful to run their own experiments in their own personal computers (e.g., overnight) they are more likely to use other more capable facilities when running large experiments. Consider for example the case of making large experiments for tuning port-folios or, of course, the case of running either one particular International Planning Competition or to arrange a new one.

A cluster with eleven computers (one front-end and ten back-end nodes) was set up for the Seventh International Planning Competition. The front end is named pleiades, while the back-end nodes are: alcyone, atlas, electra, maia, merope, taygeta, pleione, celaeno, tau and asteropeI. Each node is an Intel Xeon 2.93 Ghz Quad Core processor (64

bits) using Linux. Up to 6 GB of RAM memory and 750 GB of hard disk were available for each planner. Each planner was run in a single node and no planner was allowed to run in more than one simultaneously —although in the multicore track it was allowed to use all the 4 cores of a node. No GPU processing was available.

Nevertheless, no assumptions are made about the underlying hardware running the experiments. However, they are expected to be GNU/Linux-based computers, the reason being that the script `invokeplanner.py` uses the `/proc` filesystem to retrieve various sets of system information —see *invokeplanner.py*

2.4 Documentation

This documentation is distributed both as a pdf file and html pages. Both formats are automatically generated with Sphinx.

This documentation is distributed under the terms of the GNU General Public License 3.0 (see *GNU GENERAL PUBLIC LICENSE*) and therefore you are very welcome to modify it, copy it and/or redistribute it. If you want to do so:

1. The source files of this documentation can be retrieved with **svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/doc**
2. If you plan to modify and recompile the source files `*.rst`, then:

   2.a. Make sure to install Sphinx

   2.b. This documentation uses a wonderful contribution named blockdiag. After installing it, make sure also to make it available to Sphinx by installing and configuring sphinxcontrib-blockdiag

Now, from the root directory of your documentation type:

```
make html
```

to obtain a new version of the html pages, or:

```
make latex
```

to obtain LaTeX source files which can then be processed to obtain the final pdf. More formats are available as shown by the following command:

```
make help
```

New in version 1.2: Command-line arguments parsed with argparseNew in version 1.2: `seed.py`: flags for specifying the smtp server and port have been added; the logging services have been added; the verbose flag has been added; most of the arguments became optionalNew in version 1.2: `buildplanner.py`: logfile accepts the same placeholders than the build log file; `--ini` added; use of regular expressions instead of filename pattern expressions; the automated e-mail facility has been addedNew in version 1.2: `builddomain.py`: logfile accepts placeholders; `--ini` added; use of regular expressions instead of filename pattern expressions; the automated e-mail facility has been addedNew in version 1.2: `invokeplanner.py`: logfile accepts placeholders; `--ini` added; the verbose flag has been addedChanged in version 1.2: `invoke-planner.py`: renamed as *invokeplanner.py*New in version 1.1: `validate.py`: the step length is computed now in addition to the value returned by VAL

CHAPTER

THREE

IPCDATA

This is the first of the two important packages developed at the Seventh International Planning Competition. The second one is `IPCReport` —see *IPCReport*. A third one, with code which is particular to the IPC 2011 is distributed separately: `IPCPrivate` described in *IPCPrivate*. Besides, the `IPCTools` were developed after the IPC to ease the data analysis and it is described in *IPCTools*

The `IPCData` module consists, mainly, of six different python modules: `check.py`, `seed.py`, `buildplanner.py`, `builddomain.py`, `invokeplanner.py` and `validate.py`. All of these programs are discussed in the next sections.

3.1 Dependencies

All these modules have been developed with Python 2.x

Warning: Version 1.2 of `IPCData` uses argparse for parsing the command-line arguments. While they significantly improved the usability of the scripts, it requires either Python 2.7 or Python 3.2

The modules described in this chapter have a number of dependencies with third-party software that have to be installed prior to the installation and usage of `IPCData`:

Configobj This package provides a wrapper to read and write INI configuration files

Instructions for downloading and installing this package are given here

---

Subversion python This package provides a wrapper to access svn repositories in Python

Instructions for downloading and installing the package are given here

Finally, some modules in this package also use PrettyTable. However, since it consists of a single module (i.e., no `__init__` file is given), it is provided within the package `IPCData` by default.

A step-by-step procedure to install all these packages can be accessed in the first practical case discussed in this manual —see *First practical case*.

In what follows, it is assumed that the reader has already checked out to her local computer the scripts located at:

`svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral/IPCData`

3.2 Command-line arguments

All the modules in this package adhere to a consistent naming of the flags they acknowledge. This section describes this convention.

As a general rule all programs honour, at least, the following three flags:

-h, --help provide a brief description of the main purpose of the script and presents all the available flags

-v, --verbose shows additional information to track progress

-V, --version shows the current version of the script along with the head svn release that affects it and the building date

Besides, all modules acknowledge a number of directives for configuring the logging services:

-l, --logfile if provided, all output is redirected to the given file. It is common practice for most scripts to recognize a number of *placeholders* which are just strings starting with `$` and can be different in each case [1]. To allow the generation of logfiles with the same name, the current date and time is appended at the end. If a log filename is provided no output is shown in the standard output

-L, --level specifies the desired level of the log messages. Legal values are: `INFO`, `WARNING` and `CRITICAL`. Only messages of the same level or above are shown. In all cases, the default value is `INFO`, i.e., all messages are shown

-e, --email accepts an arbitrary number of e-mail recipients to notify upon completion of the current script. If a logfile was specified with *`--level`* then it is automatically attached to the body as well.

To configure the automated e-mail facility see *seed.py*. On the other hand, when applicable, tracks and subtracks are provided with the following directives:

-t, --track track name, e.g., `seq` or `tempo`

-s, --subtrack subtrack name such as opt or sat

In the same vain, planners and domains can be fully specified with the following arguments:

-D, --domain all domains matching any of the regular expressions given after the directive are accepted by the script

-P, --planner likewise, all planners matching any of the regular expressions given here are accepted by the script

-X, --exclude-domain all domains matching any of the regular expressions given after the directive are discarded by the script

-x, --exclude-planner as with the previous directive, all planners matching any of the regular expressions given here are discarded by the script

The next set of directives refers to other sources of information that are used by most modules in this package:

-b, --bookmark this flag specifies the svn bookmark to be used by the script. If none is given, it tries to retrieve it from the INI configuration file specified with the next argument

[1] When using the dollar sign `$` in the command line, the shell will always try to expand it to the values of environment variables. This is known as *interpolation*. To avoid it, strings containing dollar signs shall be embraced between single quotes as in '$track-$subtrack.$planner-build'

-i, --ini specify the location of an INI configuration file. If none is provided `~/.ipc.ini` is tried by default

The svn bookmark specifies the location of the svn server with the data repository —see *Contents of the data repository*. The INI configuration file is created with *seed.py*.

Finally, while most scripts accept an specification of an output directory, *validate.py* expects to get an input directory. In all cases, the directory is specified with the following directive:

-d, --directory specifies the location of a directory either for reading or writing data. In case the directory is used in write mode, specific subdirectories are created to store data and an exception is raised if they already exist

Most of these parameters accept a single value. However, a number of considerations follow:

- If two or more directives have no values, they can be abbreviated in a single argument. For example, *-vm* specifies both verbose output and test mode in the script *invokeplanner.py*
- Some directives can be specified an arbitrary number of times. For example, two e-mail recipees can be specified with *--email albert.einstein@wurttemberg.edu --email kurt.godel@vienna.edu*. However, the same command-line arguments can be abbreviated as follows *--email albert.einstein@wurttemberg.edu kurt.godel@vienna.edu*.

  Other directives that accept an arbitrary number of arguments are: *--domain*, *--planner*, *--exclude-domain* and *--exclude-planner*

3.3 check.py

Although this script is mainly thought for the organizers of an International Planning Competition, it can be used also by practitioners in their own labs. While it can be easily extended to make additional checks, it fundamentally tries to make sure that the directory where a planner resides looks like a typical directory of the IPC. Bear in mind that this script does not examine the svn repository but a local directory where a particular check out has been performed.

Planners are stored in a directory according to the following convention: planners/<*track*>-<*subtrack*>/<*track*-*subtrack*>-<*name*> where <*track*>, <*subtrack*> and <*name*> stand for the track and subtrack of a planner whose name is given in <*name*>. Besides, every directory of this type shall contain at least the **build** and **plan** scripts:

build it is a script (usually a bash script) that automatically starts the compilation of the planner when invoked. A typical content shall look like:

```
#!/bin/bash
make
```

though they can be as complex as desired

---

plan this script shall receive three parameters: *domain*, *problem* and *plan solution file* and should invoke the planner accordingly.

From the webpage of the IPC 2008, the following example has been taken: the executable for the "Find Plans Quickly" planner is called **fpq** and accepts the same command-line arguments as the FF planner. Then your plan script may look like:

```
#!/bin/bash
./fpq -o $1 -f $2
mv $2.soln $3
```

> There is only one exception to this rule. If the planner is run in `dck` mode, then it will be invoked with up to four different arguments, the fourth being the location of the *domain control knowledge* used at the Learning Track of the Seventh International Planning Competition —see *Contents of the data repository*

The script `check.py` accepts the directive `--directory` to specify a particular directory to examine. A few examples follow.

To check whether the directory where the **ArvandHerd** planner of the `seq-mco` track has been stored follows the convention of the IPC:

```
% ./check.py --directory ~/lab/ipc2011/planners/seq-mco/seq-mco-arvandherd/
------------------------------------------------------------------------
 Name      : arvandherd
 track     : sequential
 subtrack  : multicore
 signature : seq-mco-arvandherd
 directory : /Users/clinares/lab/ipc2011/planners/seq-mco/seq-mco-arvandherd/


 [1 planners found]
```

where `~/lab/ipc2011/planners/seq-mco/seq-mco-arvandherd/` is the location where this planner was checked out. The name, track and subtrack are automatically extracted from the signature which is read from the given path. Because this planner explicitly says that one planner has been found, it is also ensured that the scripts **build** and **plan** where stored at the right locations.

Also, a whole track-subtrack can be examined using the `--recursive` flag. For example, from the directory structure of the `seq-mco` track, the following check can be issued:

```
% ./check.py --directory ~/lab/ipc2011/planners/seq-mco --recursive
------------------------------------------------------------------------
 name      : acoplan
 track     : sequential
 subtrack  : multicore
 signature : seq-mco-acoplan
 directory : /Users/clinares/lab/ipc2011/planners/seq-mco/seq-mco-acoplan/

------------------------------------------------------------------------
 name      : arvandherd
 track     : sequential
 subtrack  : multicore
 signature : seq-mco-arvandherd
 directory : /Users/clinares/lab/ipc2011/planners/seq-mco/seq-mco-arvandherd/

------------------------------------------------------------------------
 ...
------------------------------------------------------------------------
 name      : yahsp2-mt
 track     : sequential
 subtrack  : multicore
 signature : seq-mco-yahsp2-mt
 directory : /Users/clinares/lab/ipc2011/planners/seq-mco/seq-mco-yahsp2-mt/


 [8 planners found]
```

Finally, the `check.py` accepts a third flag `--questionnaire` to show the pddl questionnaire of all the planners that are recognized by the script. If provided, it prints out the contents of that file.

3.4 seed.py

Originally, this script was conceived to write a lot of information to the local filesystem in the form of an INI configuration file to reuse it as much as possible, thus saving a lot of accesses to the SVN repository. In the end, however, it serves only to save a few key fields and also to make sure that access to the SVN repository is feasible.

The following argument is mandatory:

-f, --file specifies the name of the INI configuration file to create. While it is possible to provide arbitrary names to this directive, other scripts will look, by default, for the INI configuration file in `~/.ipc.ini`. However, all scripts that handle INI configuration files acknowledge a dedicated directive to override this setting.

Besides, *-b* or *--bookmark* is mandatory as well — see *Command-line arguments*. The following parameters are optional and can be provided just to qualify the current IPC:

-n, --name name of the competition currently under configuration

-p, --part every IPC is split in a number of parts. Typical parts are: deterministic, uncertainly, learning, etc. Users other than organizers of an IPC might find this useful to refer to particular sets of experiments by a label

-w, --wiki specifies the html address of the wiki used. This can be used in a future to automate uploads to the Wiki site.

-c, --cluster specifies the name of the front-end node of the cluster to be used for running all the experiments

The values of these parameters are considered to be free text and are directly copied to the preamble of the INI configuration file

The next group of arguments are required if any of the scripts of the package `IPCData` is ever expected to use the automated e-mail facility:

-m, --mailuser e-mail account to use when sending automatic e-mail messages

-M, --mailpwd password of the e-mail account. This password is not encrypted in the INI configuration file. Besides, be aware that it will be sent over the network when opening the e-mail session

-t, --smtp address of the smtp server to use when issuing automatic e-mail messages

-o, --port port of the smtp server accepting requests to send e-mails

Because all of this parameters can be easily retrieved by other users, a solution would be to use a public account. Fortunately, Google provides unrestricted access to e-mails of this type. As a matter of fact, the following account was set up for the Seventh International Planning Competition:

mailuser ipc2011.pleiades@gmail.com

mailpwd ys98NfyMenRs

smtp smtp.gmail.com

port 587

All users of this script are very welcome to use this account or to create others of their own [2].

[2] Be aware however that reusing the same google e-mail account might make others to access the results of your experiments. In case secrecy is desired, it is highly recommended to set up a different e-mail account and to provide the user and password in the INI configuration file as explained in the text.

Also, the script checks the contents of all directories in the SVN server and issues a warning if a track-subtrack is empty —maybe because there are no planner and/or because no domain and problems have been specified.

The script writes the contents of the SVN server in the INI configuration file in different sections according to the following schema: the section `[tracks]` contains the number and name of the tracks found in the SVN repository. Then, for each one it creates a new section with its name specifying the subtracks and its names. Once again, for every subtrack it creates nested subsections with the number and name of the planners and domains (if any) within that track-subtrack.

3.5 buildplanner.py

This section describes all the command-line arguments accepted by `buildplanner.py`. For a thorough discussion of the command-line arguments see *Command-line arguments*.

While planners are stored in the SVN repository, they have to be compiled locally in order to be able to run any experiments. In other words, while the source code (and other important files such as **build** and **plan**) have to be stored in the SVN repository, object code such as libraries and executables should be generated locally.

As mentioned in the preceding section (see *seed.py*) it is possible for this script (and others) to override the SVN bookmark. If a particular INI configuration file is specified with *--ini*, the svn bookmark is retrieved from the variable `svn` of the `general` section. This behaviour can be overriden if a bookmark is explicitly given with *--bookmark*. If none is provided, then the script takes the svn bookmark specified at the `~/.ipc.ini` file by default. If no INI configuration file with that name is found and no bookmark has been specified, the script issues a fatal error and exits.

Since the SVN repository can accept an arbitrary number of planners arranged in an arbitrarily large number of tracks-subtracks, this script accepts regular expressions to specify what planners are required to be compiled and which, among those, shall be excluded. The first expression is given in the mandatory flag *--planner* while the second is expliclity specified with the optional directive *--exclude-planner*. Besides, it is mandatory to provide the *--track* and *--subtrack* the planners belong to. This is useful to distinguish planners with the same name across different tracks-subtracks.

If the user does not specify any *--directory* then the planner is checked out and compiled in a subdirectory of the current working directory which is named after the planner. If a directory is given, the planner is checked out and compiled in a directory with the name of the planner under the given directory. Besides, if a log filename is given with *--logfile*, it is generated in the target directory with the detail specified with *--level* which is `INFO` by default. This command-line argument acknowledges the same placeholders than the build log file described below: *$track*, *$subtrack* and *$planner*.

Other than the log file described above, `buildplanner.py` creates also a build log file. If *--logbuild* is provided, a verbatim copy of the compilation process is stored in the given file. This is very important to make sure that the compilation went fine and that no errors were issued on the way. For example, some participants to the International Planning Competition can pose special requirements in the form of third-party software. Recording the building process and allowing those users to check the resulting files is mandatory to ensure that their code was properly compiled and linked. On the other hand, if anything ever went wrong a whole trace (with the detail provided by the compilation tools) is reported in these files. The specification of the logfile accepts placeholders which are symbolized with the dollar sign `$`. In particular, there are three recognized variables: *$track*, *$subtrack* and *$planner* which are substituted with the particular track, subtrack and name of the planner to be built. The default value of this variable is *build-$planner.log*. The build log file is left at the root where the planner was built.

Finally, it is feasible to use the automated e-mail facility provided with most scripts of this package. If *--email* is provided, all the recipients specified are sent an automatic e-mail as soon as the script ends. If a log file (specified with *--logfile*) was given then it is attached to the e-mail as well.

For example, the following command:

```
$ ./buildplanner.py --track seq --subtrack opt --planner cpt4
  --directory ~/seq-opt-cpt4 --logfile 'build.$track-$subtrack-$planner'
  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
```

checks out and compiles the planner cpt4 that entered the `seq-opt` competition [3]. Upon completion the planner is located at `~/seq-opt-cpt4/cpt4` along with a new file named `build.seq-opt-cpt4` which contains all the messages issued by the compilation and linking tools chosen by the author. A log file with the whole building process is left at `~/seq-opt-cpt4`. Besides, the following info message is shown on the terminal:

```
[Info: the following planners have been built]
[      [u'cpt4']]
```

Because all contents of this planner are checked out to the target directory, the scripts **build** and **plan** are automatically placed in the right location, the base directory of the planner. Of particular interest here is the script **build** which is automatically invoked by `buildplanner.py` to compile the selected planner according to the rules given by the authoritative author. Of course, the local computer compiling the code shall be equipped with all the dependencies specified by the planner at hand. Besides, the **build** script shall be respectful with the local computer by avoiding sudo lines, for example, or writing/modifying other parts of the local filesystem than its current tree structure.

3.6 builddomain.py

This section describes all the command-line arguments accepted by `builddomain.py`. For a thorough discussion of the command-line arguments see *Command-line arguments*.

This script has been conceived to check out the domain and problem files of a particular track-subtrack and to build the corresponding testsets. It accepts much the same directives than `buildplanner.py` (see section *buildplanner.py*) but *--logbuild*.

First of all, it is possible for this script (and others) to override the SVN bookmark. If a particular svn bookmark is specified with *--bookmark* it is used for retrieving all domain and problem files as described below. If not, the INI configuration file specified with *--ini* is used instead. If none is provided, then the script takes the svn bookmark specified at the `~/.ipc.ini` file by default. If no INI configuration file with that name is found and no bookmark has been specified, the script issues a fatal error and exits.

The domain and problems to be used shall be referenced by a combination of track and subtrack given with the flags *--track* and *--subtrack* respectively. This enables domains with the same name to be used across different tracks which is highly desirable, though sometimes it leads to some redundance when the same problems are used in different tracks [4].

The domains checked out are those specified with the mandatory flag *--domain* after excluding those, among the ones specified with the previous flag, that meet the regular expression given with the optional flag *--exclude-domain*. Both flags accept regular expressions.

Once the domains have been fully identified with their name (flags *--domain* and *--exclude-domain*) and the track and subtrack they belong to (with *--track* and *--subtrack*), the script automatically builds the testsets in a directory named after the domain beneath the target directory given in *--directory*. If none is given, `./` is used by default. Each testset is shown in a separate directory named as `testsetxxx` where xxx stands for a number in the interval 000 – 999. The contents of each particular testset are computed as follows:

[3] Note that for `buildplanner.py` to run properly there is no need to create the INI configuration file `~/.ipc.ini` if *--bookmark* is provided.

[4] For example, all problems used in the Seventh International Planning Competition in the Sequential Satisficing Multi-core track (`seq-mco`) were exactly the same ones than those used at the Sequential Satisficing track (`seq-sat`). Because the scripts developed at the IPC 2011 are always restricted to a particular combination of track and subtrack there is no other way to reuse domain and problem files other than copying them to a separate folder in the SVN repository according to the structure discussed in *Contents of the data repository*

single If the domain directory in the SVN repository (see *Contents of the data repository*) contained only a single file, it is is replicated to all the testsets along with each particular file found in the problems directory.

---

multi If the domain directory contained an arbitrary number of files, then the script makes sure that there are as many files in the domain directory as in the problems directory and creates as many testsets as files have been found. In this case, each testset consists of a pair (domain, problem) which are taken after sorting the files in both directories in ascending lexicographical order.

While the domain and problem files can be given any names in the svn repository, the files in each testset are named as `domain.pddl` and `problem.pddl` when built in the local computer.

Finally, it is possible to specify a log filename to record the whole process with *--logfile* which is left at the target directory specified with *--directory*. To discrimante among different executions of this script, the log filename specification accepts the placeholders *$track*, *$subtrack* and *$domain* which are substituted by their corresponding values. Besides, all the e-mail recipees given with *--email* are sent an automatic e-mail upon completion. If a log file is specified it is attached to the e-mail as well.

For example, the following command:

```
$ ./builddomain.py --track seq --subtrack opt --domain 'p'
--directory ~/seq-opt-pdomains --ini ~/tmp/example.ini
--logfile 'build-$domain.$track-$subtrack.log'
--email albert.einstein@wurttemberg.edu kurt.godel@vienna.edu
```

takes all the domains that appear in the `seq-opt` track in the SVN repository specified in the INI configuration file at `~/tmp/example.ini` whose name matches the regular expression `p` and creates the testsets in `~/seq-opt-pdomains`. When the process has been completed a message like the following is issued:

```
[Info: the following domains have been built]
[      [u'parcprinter', u'parking', u'pegsol']]
```

Besides, Albert Einstein and Kurt Godel will be interrupted to know that these domains have been built and an e-mail with the generated logfile will be sent to him. The logfile is named `build-p.seq-opt.log.11-11-24.20:10:36`

3.7 invokeplanner.py

This section describes all the command-line arguments accepted by `invokeplanner.py`. For a thorough discussion of the command-line arguments see *Command-line arguments*.

This script lies at the core of `IPCData`. It uses most of the different python scripts distributed with the package and also various services from the modules `buildplanner.py` and `builddomain.py`. Its importance comes from the fact that it fully automates the process of running a particular experiment which involves an arbitrary selection of planners and domains in a computer. It is expected to be, by far, the most used module in this package.

As with other scripts, `invokeplanner.py` can override the SVN bookmark specified in the `~/.ipc.ini` file (if it exists) by providing it at invocation time with *--bookmark*. If none is given, the SVN bookmark is taken from the IPC INI configuration file specified with *--ini*. If none exists and *--bookmark* has not been given, the script exits with an error.

`invokeplanner.py` is restricted to a selection of planners and domains that meet the track and subtrack provided with the mandatory directives *--track* and *--subtrack*. This is, the track and subtrack are applied both to the selection of planners and domains. As a matter of fact, this motivated the use of the flag *--test* discussed below. Since `invokeplanner.py` makes use of `build-planner.py` it also acknowledges the flags *--planner* and

--excludeplanner discussed in *buildplanner.py*. Similarly, since it silently invokes `builddomain.py` it also acknowledges the flags *--domain* and *--exclude-domain* described in *builddomain.py*. The last four arguments admit arbitrary regular expressions.

If a target directory is given with *--directory* then both the planners and domains meeting the previous expressions are checked out and built under the specified directory. If not, `./` is assumed by default. Besides, the target directory specification is used also to store all the results of the experiments.

First, `buildplanner.py` is invoked and thus, all the selected planners are checked out and built as described in *buildplanner.py*. Internally, `buildplanner.py` is invoked with a logfile which is named after the substitutions in *build-$planner.log*. Besides, it can also generate a log file which records the results of the whole process. This can be specified with *--logfile* and it recognizes the following placeholders *track*, *subtrack*, *planner* and *domain*. This logfile can record a lot of information at various levels, such as the current version, release and built date of the script; parameters provided to the script (for the sake of completeness); the particular steps followed for building the planners and domains and the time instants when each planner was invoked with every testset. It finally ends with various pretty ASCII tables (created with PrettyTable.py) that show the overall running time, the overall memory used, the number of problems solved and the number of plan solution files generated by each planner in each domain. Each line shown in this logfile is accompanied by a number of fields that include the current local date and time, the username and hostname where the execution was performed, the name of the script and Python function that issued the message and a string with the message level. The message level can be specified with *--level* and should be one among `INFO`, `WARNING` and `CRITICAL`. Only messages with a level higher or equal than the one specified are printed out.

After `buildplanner.py` is over, the planners are located under the target directory specified with *--directory*.

Next, the testsets for the domains that satisfy the regular expressions given in *--domain* and *--exclude-domain* are created. Because it might be easily the case that one might want to run some experiments with some preliminary problems instead of using the official ones, an additional flag *--test* is acknowledged. If *--test* (with no arguments) is provided, domains are selected from the directory <`t` *track* – *subtrack*> whereas the planners are checked out from the directory with the same name without the heading `t`. This avoids the additional burden to create ancilliary directories which simply replicate the planners with strange names when all that is needed is just to run planners with testing domains. This is very useful, for example, to get an overall idea of the difficulty of some domains (especially when they are new) prior to the selection of the definitive problems.

As soon as `buildomain.py` is over, the testsets are located under the target directory given with *--directory*.

If *--email* is used, the specified recipees are sent an e-mail upon completion of the algorithm. Besides, if a log file was specified with *--logfile* then it is attached to the e-mail as well.

At last, but not least, each planner is allotted a fixed amount of time and space for solving each task. If it does not, it (and all its children) are automatically killed by the script. These limits are set with the directives *--timeout* (in seconds) and *--memory* (in Gigabytes). For example, at the Seventh International Planning Competition the bounds used were *--timeout 1800 --memory 6*. A couple of notes about these limits follow:

- The time is measured as computation time for the planner and all its subprocess/threads. Therefore, the wall-clock time shall be usually expected to be slightly larger than the time specified with the directive *--timeout*
- The memory allotted to the planner is, from the Python documentation pages: *The maximum area of address space which may be taken by the process*. In particular, planners creating other subprocesses or threads (take for example planners from the Sequential Satisficing Multi-core Competition or portfolios in any track) are restricted to do not reach this limit in total. This is, taking also into consideration the space taken by all the other suprocess and threads launched by the same planner. In the end, it is the author's responsibility of each planner to guarantee that the limit is not reached. If so, the planner (and all its children) are automatically killed by `invokeplanner.py`.

Once all the parameters have been set, `invokeplanner.py` invokes the corresponding script **plan** of each selected planner. For a full discussion of the design and purpose of this script see *check.py*.

During the execution, the script records the values of a number of parameters:

- The total time the planner is taking (taking also into account that of their children) at rather regular intervals of five seconds.
- The total memory the planner is taking (taking also into account that of their children) at rather regular intervals of five seconds.
- The number of processes currently running at rather regular intervals of five seconds.
- The number of threads currently running at rather regular intervals of five seconds.
- The number of solutions found at the end, if any.
- The time elapsed since the beginning when each solution was found.
- The overall running time (in clock-wall time) consumed by the planner.
- The maximum memory consumed by the planner during the whole execution.
- The memory the planner was taking just before voluntarily resuming execution or being killed. Note that this is not necessarily equal to the maximum memory —though this is usually the case. While most planners take memory incrementally, some take and release memory so that they show a memory profile which decays from time to time.

All this information is stored at a separate tree structure rooted at the target directory specified with *--directory* with the name `results/` —for a full discussion of the contents of this directory see *The results directory*. Beneath this directory, a number of subdirectories with the names of all the planners just built are shown. Under each one, there are as many folders as domains have been built with `builddomain.py`. Finally, each of these directories contain as many subdirectories as problems have been generated whose name meet the format xxx where xxx is an index in the range 000–999. For example, the results of the execution of the Sequential Optimal planner **bjolp** when faced with the first problem of the domain floortile are stored in `results/bjolp/floortile/000`. The following paragraphs describe the contents of each of these directories.

The previous fields are recorded at a logfile, whose name results from a combination of the track, subtrack, planner, domain and problem and looks like `_set-opt.bjolp-floortile.000-log` —for the case of the first problem of the floortile domain when solved by the Sequential Optimal planner **bjolp**. For illustration purposes, their contents are shown below:

```
[Fri, 05-20-11 02:57:55.045234]

Current working directory: /home/plg/seq-opt-floortile/_bjolp.floortile.000
Timeout: 1800 seconds
Memory : 6442450944 bytes

[real-time 5] total_time: 4.92
[real-time 5] total_vsize: 150.93
[real-time 5] num_processes: 4
[real-time 5] num_threads: 4
Number of solutions found: 1
Time/solution            : [8]
Overall runtime: 10 seconds
Overall memory : 150.93 Mbytes
Maximum memory : 150.93 Mbytes

[Fri, 05-20-11 02:58:05.093734]
```

Other information recorded in the form of files is the following (the names of the files actually created in our running example are given for the sake of clarity):

`_seq-opt.bjolp-floortile.000-cpu`

It shows information on the CPUs used as provided by the `/proc` Linux filesystem.

An example follows [5]:

```
[Fri, 05-20-11 02:57:55.045340]

* CPUinfo:

processor        : 0
vendor_id        : GenuineIntel
cpu family       : 6
model            : 30
model name       : Intel(R) Xeon(R) CPU           X3470  @ 2.93GHz
stepping         : 5
cpu MHz          : 1200.000
cache size       : 8192 KB
physical id      : 0
siblings         : 8
core id          : 0
cpu cores        : 4
apicid           : 0
initial apicid   : 0
fpu              : yes
fpu_exception    : yes
cpuid level      : 11
wp               : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                   pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
                   syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
                   xtopology nonstop_tsc aperfmperf pni dtes64 monitor ds_cpl vmx
                   smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm
                   ida dts tpr_shadow vnmi flexpriority ept vpid
bogomips         : 5866.76
clflush size     : 64
cache_alignment  : 64
address sizes    : 36 bits physical, 48 bits virtual
power management :

 ...

[Fri, 05-20-11 02:57:55.045701]
```

`_seq-opt.bjolp-floortile.000-mem`

It shows various fields about the available memory and its usage as provided by the `/proc` Linux filesystem.

The contents of this file are shown below:

```
[Fri, 05-20-11 02:57:55.045372]

* MEMinfo:

MemTotal:        8187144 kB
MemFree:         5520668 kB
Buffers:           48256 kB
Cached:          2287552 kB
SwapCached:            0 kB
Active:          2018184 kB
```

[5] Incidentally, one of the flags of our processors is ida. I wonder whether Intel already implemented the Iterative-deepening A* in their computers :)

```
Inactive:         339268 kB
Active(anon):      22096 kB
Inactive(anon):      120 kB
Active(file):    1996088 kB
Inactive(file):   339148 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:      43163644 kB
SwapFree:       43163644 kB
Dirty:            254940 kB
Writeback:             0 kB
AnonPages:         22012 kB
Mapped:            10832 kB
Shmem:               236 kB
Slab:              108080 kB
SReclaimable:      94576 kB
SUnreclaim:        13504 kB
KernelStack:        1928 kB
PageTables:         1828 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    47257216 kB
Committed_AS:      79012 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      299208 kB
VmallocChunk:   34359435848 kB
HardwareCorrupted:     0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:        7744 kB
DirectMap2M:     8372224 kB

[Fri, 05-20-11 02:57:55.045725]
```

`_set-opt.bjolp-floortile.000-ver`

It shows the current version of Linux as provided by the `/proc` Linux filesystem.

An example (properly intended to fit the page width) follows:

```
[Fri, 05-20-11 02:57:55.045305]

* Version: ['Linux version 2.6.35-23-generic (buildd@allspice)
  (gcc version 4.4.5 (Ubuntu/Linaro 4.4.4-14ubuntu5) )
  #41-Ubuntu SMP Wed Nov 24 11:55:36 UTC 2010\n']

[Fri, 05-20-11 02:57:55.045672]
```

Besides, `invokeplanner.py` captures both the standard error and the standard output and stores everything in the files `planner.err` and `planner.log` respectively. These files are useful for diagnoses purposes and attention shall be given to them by the authors of planners to ensure that they write useful data.

Finally, other files stored with all of these are the `domain.pddl`, and `problem.pddl` picked up and any collection of plan solution files generated by the planner with the syntax `plan.soln.n` where *n* stands for a number that increases from 1 in steps of 1; also, the logfile generated by `buildplanner.py` is copied here for the sake of

completeness so that each one of these directories contains information enough to be analyzed on their own, without further dependencies.

To end, a couple of examples follow:

Case 1 In this example, the planners **lmtd**, **yahsp2** and **yahsp2-mt** from the Temporal Satisficing track are going to be invoked over all problems of the domains crewplanning and sokoban. The results shall be automatically sent both to albert.einstein@wurttemberg.edu and kurt.godel@vienna.edu. A human readable recording of all events shall be directed to a logfile named `example-1`. In this case, each planner will be given only 15 minutes and 2 Gb of memory:

```
$ ./invokeplanner.py --track tempo --subtrack sat
  --planner "lmtd|yahsp.*" --domain "crewplanning|sokoban"
  --directory ~/case-1 --logfile case-1
  --email albert.einstein@wurttemberg.edu kurt.godel@vienna.edu
  --timeout 900 --memory 2
```

Case 2 The following example just runs the whole Seventh International Planning Competition of the Sequential Satisficing track when using the right SVN bookmark (which is provided as well):

```
$ ./invokeplanner.py --track seq --subtrack sat --planner ".*"
  --domain ".*" --directory ~/ipc-2011 --logfile my-ipc-2011
  --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
  --timeout 1800 --memory 6
```

Usually, the `results/` directory takes a lot of space. Therefore, a good idea might be to copy most of its structure to a separate directory preserving only the `domain.pddl`, `problem.pddl`, the plan solution files generated and the log file generated by `invokeplanner.py`. While the original contents of the `results/` directory were named `raw/` in the context of the Seventh International Planning Competition, this additional copy was known as `val/` since it contains the essential information to be validated thanks to the next script.

3.8 validate.py

A mandatory step not only at the International Planning Competition but also at any experiments run at the home lab is to validate the solutions. To this end, VAL has been used. The latest version at the time of the Seventh International Planning Competition was 4.2.08. However, version 1.1 uses version 4.2.09. A copy of it can be retrieved from http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software

Since the number of plan solution files might be huge, the validation process has been also automated. The script `validate.py` takes care of it. It receives a single mandatory parameter *`--directory`* which specifies the current location of a particular case. If the given directory contains a `domain.pddl`, `problem.pddl` and various plan solution files, these are given to VAL and its output is recorded in a separate log file which is named after the track, subtrack, planner, domain and problem at hand which looks like `_seq-opt.bjolp-floortile.000-val`. If the given directory does not contain these files but there are other subdirectories, the entire tree structure is recursively traversed validating all the cases found along the path.

Besides, if a log file is specified with *`--logfile`* then messages issued at the specified *`--level`* or above are recorded. In case that one or more e-mail recipees are given with *`--email`*, they are all notified upon completion of the script. Note that to enable the automated e-mail facility, it is mandatory to provide an INI configuration file with *`--ini`* unless `~/.ipc.ini` shall be used.

Henceforth, if all the results of a particular invocation of `invokeplanner.py` are stored at the directory `~/my-ipc-2011`, the following command validates all the solutions generated so far:

```
$ ./validate.py --directory ~/my-ipc-2011
```

and leaves a validation log file at each location where it found the essential files to perform the validation. In our running example, the file _seq-opt.bjolp-floortile.000-val contains the following data:

```
[Thu, 10-06-11 00:11:13.276711]

Solution file: plan.soln
Size         : 892
Status       : 1
Value        : 38
Step length  : 25
return code  : 0
stderr       : []

Number of solution files found: 1
Number of correct solutions found: 1

[Thu, 10-06-11 00:11:13.313025]
```

The fields shown are self-explanatory. In case of error, the corresponding validation log file shows the appropriate data [6]. New in version 1.3: `test.py` has been created and thus, a new requirement on third-software is posted: NumPy and SciPy.New in version 1.2: Command-line arguments parsed with argparseNew in version 1.1: `report.py` accepts a new collection of variablesNew in version 1.1: `score.py` accepts now a new command-line argument *--time*New in version 1.1: Welcome `tscore.py`!

[6] Note that from version 1.1 the validation log contains a new line with the step length of each plan

CHAPTER

FOUR

IPCREPORT

While the design and implementation of the package `IPCData` (see *IPCData*) followed much the ideas that were already developed at the Sixth International Planning Competition in 2008, this package is brand new. It is mainly devoted to provide some simple (yet hopefully useful) mechanisms to access the data generated during the IPC or, alternatively, during a number of experiments.

The `IPCReport` package consists mainly of three different Python modules: `report.py`, `score.py` and `tscore.py`. While the first is intended to inspect the data generated by the `invokeplanner.py` module (see *invokeplanner.py*), the second and third have been developed to provide a consistent way to compute score tables and serves to compare the performance of a selected subset of planners in a selected subset of domains.

All these modules have been developed with Python 2.x

4.1 Dependencies

The modules described in this chapter have a number of dependencies with third-party software that have to be installed prior to the installation and usage of `IPCReport`:

pyExcelerator This package provides an easy-to-use and clean interface to the generation of Excel worksheets with a number of nice features including colors, splitters, etc.

Instructions for downloading and installing the package are given here

numPy and SciPy NumPy is, according to its authors, the fundamental package for scientific computing with Python. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation for different purposes.

Instructions to install these packages are given here

Finally, PrettyTable is used also. However, since it consists of a single module (i.e., no `__init__` file is given), it is provided within the package `IPCReport` by default.

In what follows, it is assumed that the reader has already checked out to her local computer the scripts located at:

```
svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral/IPCReport
```

4.2 Command-line arguments

All the modules in this package adhere to a consistent naming of the flags they acknowledge. This section describes this convention.

As a general rule all programs honour, at least, the following three flags:

-h, --help	provide a brief description of the main purpose of the script and presents all the available flags
-q, --quiet	only prints the requested data
-V, --version	shows the current version of the script along with the head svn release that affects it and the building date

All the modules in package `IPCReport` can process data either from a results tree directory (see *The results directory*) or a summary —also known as snapshot, see *Snapshots*. While they cannot be specified simultaneously, one has to be provided with one of the following command-line arguments:

-d, --directory	specifies the directory to explore. Their contents have to be consistent with the structure of the results directory, see *The results directory*
-s, --summary	instructs the script to retrieve the data contained in the binary file specified. For more information see *Snapshots*

On the other hand, all modules provide simple means to filter data by planner, domain and problem. In all cases, the given command-line argument receives a regular expression and only one:

-P, --planner	only planners meeting the specified regexp are considered. All by default
-D, --domain	only domains meeting the specified regexp are considered. All by default
-I, --problem	only problem ids meeting the specified regexp are examined. All by default

The output of every module consists always of a table (with different sorts of data, according to the purpose of the module) that can be generated in different formats and can be given arbitrary names:

-n, --name	name of the output table. In some cases, the name can acknowledge placeholders
-y, --style	sets the table type. At least, `table`, `octave`, `html`, `excel` and `wiki` are honoured by all modules. Exceptionally, `score.py` also welcomes `latex`

Most of these parameters accept a single value. However, some directives can be specified an arbitrary number of times. For example, one might want to examine the contents of the variables `solved` and `oksolved` with `report.py`. Instead of writing *`--variable solved --variable oksolved`*, it is possible to abbreviate it as *`--variable solved oksolved`*. Other directives that accept an arbitrary number of arguments are: *`--ascending`* and *`--descending`*

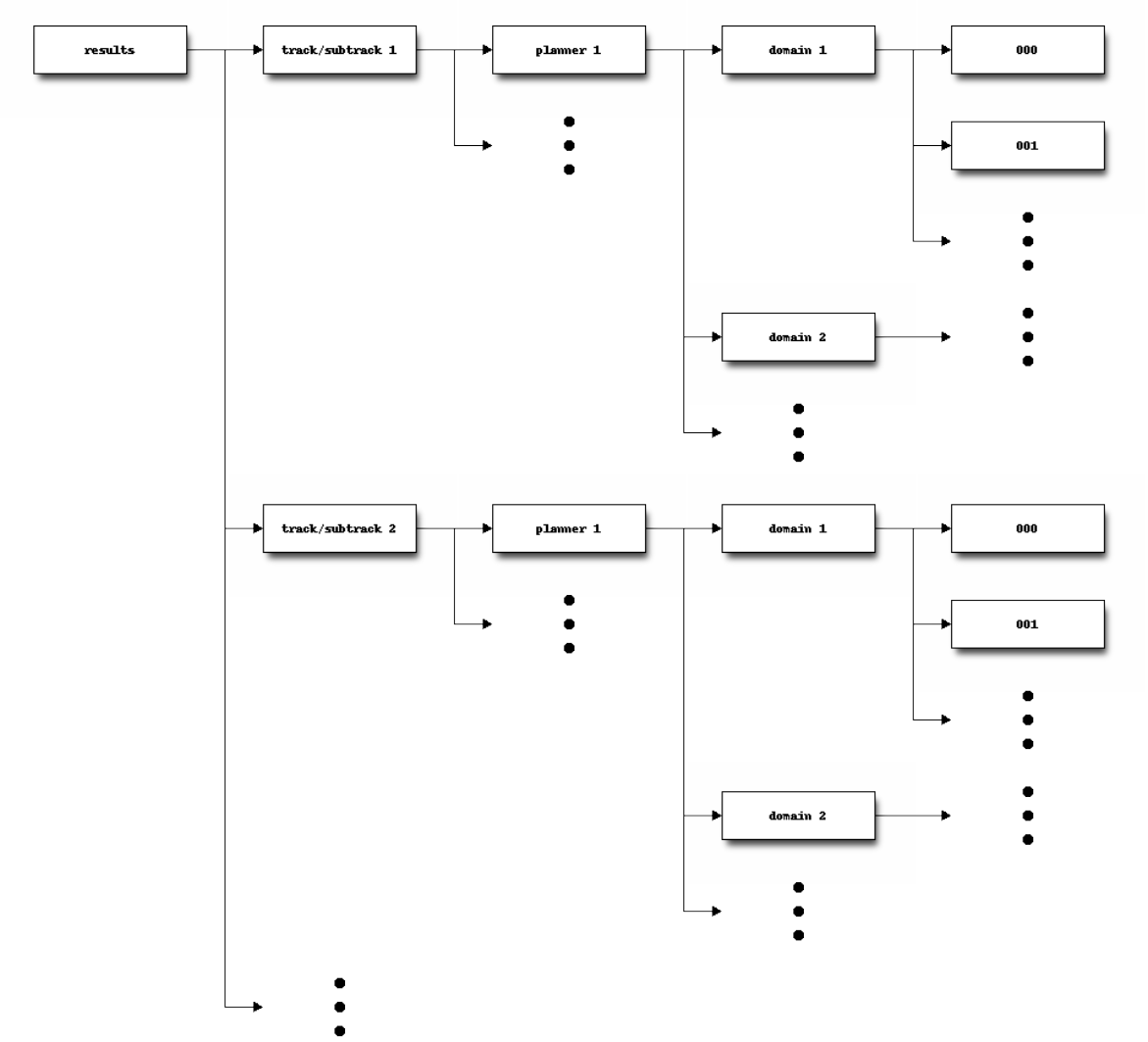
4.3 report.py

`invokeplanner.py` generates a particular tree structure that starts at the directory `results/` in the same directory specified with the command-line option *`--directory`*. All modules of the package `IPCReport` are able to process the data in a results directory. However, this might result in large waiting times. To speed up the process, summaries (alternatively known also as snapshots) are provided.

While there is no need to be aware of the particular arrangement of the results directory structure it is described here succintly for the sake of completeness. Also, a gentle introduction to snapshots is provided immediately after. Most readers can safely skip the first two subsections and go directly to the next subsection that explains how to inspect data, *Inspecting data*.

4.3.1 The results directory

The contents of the `results/` directory are sketched below:



Therefore, the particular results of executing planner *P* in domain *D* in a particular track/subtrack are all stored in a number of directories `000/`, `001/`, ... To examine the results of one execution it is enough to examine the contents of that particular directory. Recall that these directories contain the problems and domains stored in the svn repository that result after sorting the names of domains (if more than one) and problems in lexicographical order —according to the cases `single` and `multi`, see *builddomain.py*.

In particular, the contents of the results directory of the Seventh International Planning Competition can be accessed in:

`svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/results`

This repository contains three subdirectories:

logs/

This directory contains the execution log files that were generated by the script `invokeplanner.py` and also the standard output generated by the script itself. The first one refers to the logfile generated

with the directive *–logfile*, and the second one is just the standard `nohup.out` output that results when running a process in background that is explicitly requested to do not be a child of the current shell process —explicitly requested with the **/usr/bin/nohup** command that automatically records the standard output to the file `nohup.out`.

For the sake of clarity, a chunk of the output of the log file specified with *`--logfile`* is shown here that shows the result of running all planners in the `seq-opt` competition with the `sokoban` domain:

```
[2011-03-30 12:55:49,375] [       plg@tau] [invoke_planner::show_switches] INFO

------------------------------------------------------------------------------
 * Track     : 'seq'
 * Subtrack  : 'opt'
 * Planner   :  ['*']
 * Domain    :  ['sokoban']

 * Directory :  /home/plg/seq-opt-sokoban
 * Bookmark  :  svn+ssh://svn@korf.plg.inf.uc3m.es/ipc2011

 * Timeout   :  1800 seconds
 * Memory    :  6442450944 bytes
 ------------------------------------------------------------------------------

[2011-03-30 12:55:49,376] [       plg@tau] [invoke_planner::setup] INFO
  Building planner ...
[2011-03-30 12:55:50,209] [       plg@tau] [co_and_build] INFO
         Checking out bjolp in '/home/plg/seq-opt-sokoban/bjolp'
[2011-03-30 12:55:51,403] [       plg@tau] [buildplanner:co_and_build] INFO
         Building bjolp
[2011-03-30 12:59:15,735] [       plg@tau] [co_and_build] INFO
         Checking out cpt4 in '/home/plg/seq-opt-sokoban/cpt4'
[2011-03-30 12:59:21,068] [       plg@tau] [buildplanner:co_and_build] INFO
         Building cpt4

...

[2011-03-30 13:36:52,066] [       plg@tau] [invoke_planner::setup] INFO
  Building domain ...
[2011-03-30 13:36:52,735] [       plg@tau] [build_domain] INFO
         Building domain sokoban in '/home/plg/seq-opt-sokoban/sokoban'
[2011-03-30 13:37:01,694] [       plg@tau] [invoke_planner::setup] INFO
  Building workingdir /home/plg/seq-opt-sokoban/_bjolp.sokoban.000 ...
[2011-03-30 13:37:07,015] [       plg@tau] [invoke_planner::collect] INFO
  Collecting results in /home/plg/seq-opt-sokoban/_bjolp.sokoban.000 ...
[2011-03-30 13:37:07,064] [       plg@tau] [invoke_planner::setup] INFO
  Building workingdir /home/plg/seq-opt-sokoban/_bjolp.sokoban.001 ...

...

[2011-03-31 01:47:46,860] [       plg@tau] [invoke_planner::show_stats] INFO

 * Overall running time (seconds):
+------------------+---------------+---------------+
| *                |    sokoban    |     total     |
+------------------+---------------+---------------+
| bjolp            | 1078.88212848 | 1078.88212848 |
| cpt4             | 12922.8750858 | 12922.8750858 |
| fd-autotune      | 887.525495529 | 887.525495529 |
| fdss-1           |  616.94525528 |  616.94525528 |
```

```
| fdss-2           | 611.941836357 | 611.941836357 |
| forkinit         | 3423.95640659 | 3423.95640659 |
| gamer            | 17898.9191561 | 17898.9191561 |
| iforkinit        | 216.029652119 | 216.029652119 |
| lmcut            | 878.142556906 | 878.142556906 |
| lmfork           | 4155.72897196 | 4155.72897196 |
| merge-and-shrink | 611.936996937 | 611.936996937 |
| selmax           |  456.77576685 |  456.77576685 |
| total            | 43759.6593089 |               |
+------------------+---------------+---------------+

[2011-03-31 01:47:46,861] [       plg@tau] [invoke_planner::show_stats] INFO

 * Overall memory (Mbytes):
+------------------+---------------+---------------+
| *                |    sokoban    |     total     |
+------------------+---------------+---------------+
| bjolp            | 11903.9882812 | 11903.9882812 |
| cpt4             |  740.85546875 |  740.85546875 |
| fd-autotune      |  714.7890625  |  714.7890625  |
| fdss-1           |   10267.1875  |   10267.1875  |
| fdss-2           | 11388.7304688 | 11388.7304688 |
| forkinit         |  430.98046875 |  430.98046875 |
| gamer            | 63276.8867188 | 63276.8867188 |
| iforkinit        |   366.71875   |   366.71875   |
| lmcut            |  703.5390625  |  703.5390625  |
| lmfork           |  450.95703125 |  450.95703125 |
| merge-and-shrink | 11436.7890625 | 11436.7890625 |
| selmax           | 1637.82421875 | 1637.82421875 |
| total            | 113319.246094 |               |
+------------------+---------------+---------------+

[2011-03-31 01:47:46,862] [       plg@tau] [invoke_planner::show_stats] INFO

 * Number of solved instances:
+------------------+---------+-------+
| *                | sokoban | total |
+------------------+---------+-------+
| bjolp            |    20   |   20  |
| cpt4             |    1    |   1   |
| fd-autotune      |    20   |   20  |
| fdss-1           |    20   |   20  |
| fdss-2           |    20   |   20  |
| forkinit         |    19   |   19  |
| gamer            |    19   |   19  |
| iforkinit        |    20   |   20  |
| lmcut            |    20   |   20  |
| lmfork           |    19   |   19  |
| merge-and-shrink |    20   |   20  |
| selmax           |    20   |   20  |
| total            |   218   |       |
+------------------+---------+-------+

[2011-03-31 01:47:46,863] [       plg@tau] [invoke_planner::show_stats] INFO

 * Number of overall solutions generated:
+------------------+---------+-------+
| *                | sokoban | total |
```

```
+------------------+---------+-------+
| bjolp            |    20   |   20  |
| cpt4             |    1    |   1   |
| fd-autotune      |    20   |   20  |
| fdss-1           |    20   |   20  |
| fdss-2           |    20   |   20  |
| forkinit         |    19   |   19  |
| gamer            |    19   |   19  |
| iforkinit        |    20   |   20  |
| lmcut            |    20   |   20  |
| lmfork           |    19   |   19  |
| merge-and-shrink |    20   |   20  |
| selmax           |    20   |   20  |
| total            |   218   |       |
+------------------+---------+-------+
```

The dump shown above is divided in four sections: the first one shows some administrative information with the current version of the script along with a description of all the parameters given to it. Next, various `INFO` messages are issued to show what planners have been checked out from the svn repository and the exact times when they were compiled; the third part shows the testsets built; finally, a human-readable output is shown with some statistics about the overall performance of all planners.

On the other hand, the output of the **nohup** command is shown below:

```
Revision: 115
Date: 2011-03-27 16:08:59 +0200 (Sun, 27 Mar 2011)

./invokeplanner.py 1.0
```

There is no particular arrangement for the contents of the `logs` directory though the most usual is to have first a number of subdirectories sorted by track and subtrack. Beneath these directories different folders exist, each referring to a particular set of experiments. For example, a directory named `acoplan` means that it contains the output of running the planner **acoplan** with all domains. A subdirectory named `barman` shall be expected to contain the results of all planners when facing that particular domain, and so on.

raw/

This directory contains the tree structure that results after merging the directory `results/` of all the experiments performed so far in all tracks.

However, none of these directories contain solutions validated by the Automatic Validation Tool VAL

val/

This directory follows the same structure than the directory `raw/` but it contains only the minimum number of files that are necessary for validating each solution —if any was generated.

The script `validate.py` was later run on this directory, leaving a validation log file at each terminal directory with the result of the validation process. For more details, the interested reader is referred to *validate.py*

Therefore, from the previous descriptions it follows that all the results of the Seventh International Planning Competition are available in two different formats: either raw or validated. Unfortunately, processing these directories takes usually a long time. To speed it up snapshots are provided.

4.3.2 Snapshots

A snapshot (or alternatively, a summary) is just a binary file that contains the same relevant data stored in a results tree directory. Besides, it follows the same structure depicted there.

Snapshots provide a number of advantages:

Speed handling a binary file is far faster than traversing a tree structure, visiting files and parsing their contents

Size besides, snapshots are usually smaller than a compressed file with the contents of a results directory so that they ease exchanging data among developers or the participants/organizers of an International Planning Competition

Snapshots are just created by instructing `report.py` to write the result of a query in a file specified with the directive `--summarize`, i.e., snapshots contain the results that were processed from a particular directory or another snapshot. In the first case, the tree that contains the data to inspect is specified with the directive `--directory` whereas a snapshot can be specified with the flag `--summary` —see *Command-line arguments*.

While they contain all the necessary information to understand and analyze the performance of each planner in every single domain, they are not easy to process manually. Instead, a dedicated module is devoted to this goal: `report.py`

4.3.3 Inspecting data

This section describes all the command-line arguments accepted by `report.py`. For a thorough discussion of the command-line arguments see *Command-line arguments*. Besides, the data retrieved with the script described here can be directly given to `test.py` (see *test.py*) to perform various sorts of statistical tests.

The `report.py` scripts accepts trees as the one described in subsection *The results directory* with the option `--directory`. Besides, it can also accept a binary file with the same contents, termed here as `summaries` or `snapshots` (see section *Snapshots*) with the directive `--summary`. In the following we will refer both to the snapshots and the results tree directories as the *origin*.

The specified origin automatically sets the *level* of the query. If it refers to a directory that looks like `track-n-subtrack-m` (such as `seq-sat`), the query refers to the whole track/subtrack; if the origin is relative to a directory such as `planner-p` (such as `cbp`), the queries are referred only to that particular planner in the track/subtrack that contains it; if the origin points deeper to `domain-d` (e.g., `tidybot`) then all queries are relative to the combination of planner and domain that are defined within that particular track/subtrack. Finally, specifying an origin with a particular problem restricts all queries to that particular problem. However, the level set by default by a particular origin can be altered with `--level`. Both snapshots and results tree directories are arranged as explained in *The results directory*. Therefore, the only legal levels are: `planner`, `domain` and `problem` and in exactly that order. Obviously, the level cannot be pushed up (e.g., involving other planners when specifying a domain) but it can be refined further by specifying any of the legal levels if and only if the origin is the same or above than the specified level.

Furthermore, queries can be refined by a number of arguments, as explained in *Command-line arguments* by providing one (and only one) regular expression to any of the following directives: `--planner`, `--domain` and/or `--problem`. They can be provided in any combination. For example, `--planner lama --domain p --problem "0[01][02468]"` retrieves information for the problems with an even identifier in those domains that start with *p* that where given to planners whose name starts with *lama*.

`report.py` acknowledges a number of variables whose values are returned after inspecting the corresponding origin. The list of available variables is listed if `--variables` is specified in the command line —for a thorough introduction to the variables acknowledged by `report.py` the reader is referred to *Reporting variables*. Variables are specified with the directive `--variable`. There is no need to use the directive more than once unless the specification of different variables happen in different locations of the command-line. For example, to access vars *var1*, *var2*, *var3*, ... the following suffices: `--variable var1 var2 var3 ...`. The report will show the variables

in the same order they have been specified so that the same results can be achieved with *--variable var3 var2 var1 ...* but in a different order.

For example, the number of problems in the sequential satisficing track of the Seventh International Planning Competition can be retrieved with the following command:

```
$ ./report.py --directory /Volumes/Owl/Downloads/ipc2011/results/val/seq-sat
  --variable numprobs --summarize seq-sat.snapshot

Revision: 282
Date: 2011-07-04 10:48:49 +0200 (Mon, 04 Jul 2011)

./report.py 1.0

--------------------------------------------------------------------------------
* directory      : /Volumes/Owl/Downloads/ipc2011/results/val/seq-sat
* snapshot       : /Users/clinares/lab/ipc2011-data/scripts/pycentral/IPCReport/seq-sat.snapshot
* name           : report
* level          : None
* planner        : .*
* domain         : .*
* problem        : .*
* variables      : ['numprobs']
* unroll         : False
* sorting        : []
* style          : table
--------------------------------------------------------------------------------

name: report
+----------+
| numprobs |
+----------+
|   7560   |
+----------+
legend:
  numprobs: total number of problems [elaborated data]

created by IPCrun 1.0 (Revision: 283), Thu Jul 21 13:42:08 2011
```

Note that the preceding command creates also a summary with all the data that results from processing all the tree structure rooted at the particular location given `/Volumes/Owl/Downloads/ipc2011/results/val/seq-sat`

The same query can be refined further requesting the number of problems by planner just by altering the level as follows:

```
$ ./report.py --summary seq-sat.snapshot --variable numprobs --level planner

Revision: 282
Date: 2011-07-04 10:48:49 +0200 (Mon, 04 Jul 2011)

./report.py 1.0

--------------------------------------------------------------------------------
* summary        : /Users/clinares/lab/ipc2011-data/scripts/pycentral/IPCReport/seq-sat.snapshot
* snapshot       :
* name           : report
* level          : planner
* planner        : .*
* domain         : .*
```

```
 * problem        : .*
 * variables      : ['numprobs']
 * unroll         : False
 * sorting        : []
 * style          : table
 ------------------------------------------------------------------------------

 name: report
+---------------+----------+
|    planner    | numprobs |
+---------------+----------+
|    acoplan    |   280    |
|    acoplan2   |   280    |
|     arvand    |   280    |
|      brt      |   280    |
|      cbp      |   280    |
|      cbp2     |   280    |
|      cpt4     |   280    |
|   dae_yahsp   |   280    |
| fd-autotune-1 |   280    |
| fd-autotune-2 |   280    |
|     fdss-1    |   280    |
|     fdss-2    |   280    |
|  forkuniform  |   280    |
|   lama-2008   |   280    |
|   lama-2011   |   280    |
|     lamar     |   280    |
|     lprpgp    |   280    |
|   madagascar  |   280    |
|  madagascar-p |   280    |
|     popf2     |   280    |
|     probe     |   280    |
|    randward   |   280    |
|     roamer    |   280    |
|  satplanlm-c  |   280    |
|    sharaabi   |   280    |
|     yahsp2    |   280    |
|   yahsp2-mt   |   280    |
+---------------+----------+
 legend:
   planner [key]
   numprobs: total number of problems [elaborated data]

 created by IPCrun 1.0 (Revision: 283), Thu Jul 21 13:43:36 2011
```

Note that because a snapshot was created in the first query, it is now feasible to use it instead of directly accessing the tree structure. This procedure actually saves a lot of time and goes far faster.

Of course, variables can be combined. For example, the following command returns the number of problems, the number of solved tasks (but not validated) and the number of problems where the winners of the Sixth International Planning Competition (LAMA 2008) and the Seventh International Planning Competition (LAMA 2011) failed:

```
 $ /report.py --summary seq-sat.snapshot --planner 'lama.*20.*' --level planner
   --variable numprobs numsolved numfails --quiet

 name: report
+-----------+----------+-----------+----------+
|  planner  | numprobs | numsolved | numfails |
+-----------+----------+-----------+----------+
```

```
| lama-2008 |   280    |    188    |    92    |
| lama-2011 |   280    |    250    |    30    |
+-----------+----------+-----------+----------+
 legend:
   planner [key]
   numprobs: total number of problems [elaborated data]
   numsolved: number of solved problems (independently of the solution files generated) [elaborated d
   numfails: total number of fails [elaborated data]

 created by IPCrun 1.0 (Revision: 283), Thu Jul 21 13:51:06 2011
```

In this case, because the directive *--quiet* was given, all the headers were removed from the output.

Moreover, the results can be sorted either in ascending or descending order of any combination of variables thanks to the flags *--ascending* and *--descending*. These variables shall be given along with one of the variables specified in the query and/or any of the legal levels: `planner`, `domain` and/or `problem`. For example, the following command shows the number of problems successfully solved and the number of plan solution files generated by all planners in the sequential satisficing track. It then sorts the output giving preference to the planners that solved more tasks and, in case of a tie (note the case of planners **probe** and **fdss-2** both with 233 tasks solved), it ranks first those that generated more solution files:

```
$ ./report.py --summary seq-sat.snapshot --variable oknumsolved oksumnumsols
  --level planner --quiet --descending oknumsolved --descending oksumnumsols
 name: report
+---------------+-------------+--------------+
|    planner    | oknumsolved | oksumnumsols |
+---------------+-------------+--------------+
|   lama-2011   |     250     |     874      |
|     fdss-2    |     233     |     645      |
|     probe     |     233     |     460      |
|     fdss-1    |     232     |     828      |
| fd-autotune-1 |     223     |     557      |
|     roamer    |     213     |     779      |
|  forkuniform  |     207     |     589      |
|     lamar     |     195     |     764      |
| fd-autotune-2 |     193     |     516      |
|     arvand    |     190     |     1813     |
|   lama-2008   |     188     |     743      |
|    randward   |     184     |     689      |
|      brt      |     157     |     499      |
|     yahsp2    |     138     |     246      |
|   yahsp2-mt   |     137     |     423      |
|      cbp2     |     135     |     834      |
|      cbp      |     123     |     788      |
|   dae_yahsp   |     120     |     963      |
|     lprpgp    |     118     |     236      |
|  madagascar-p |      88     |      88      |
|     popf2     |      81     |     100      |
|   madagascar  |      67     |      67      |
|      cpt4     |      52     |      52      |
|    sharaabi   |      33     |      33      |
|  satplanlm-c  |      32     |      32      |
|    acoplan    |      20     |      80      |
|    acoplan2   |      20     |      70      |
+---------------+-------------+--------------+
 legend:
   planner [key]
   oknumsolved: number of *successfully* solved problems (independently of the solution files generat
```

```
                [elaborated data]
   oksumnumsols: sum of the total number of *successful* solution files generated [elaborated data]

 created by IPCrun 1.0 (Revision: 283), Thu Jul 21 14:09:09 2011
```

Another very interesting flag is *--unroll*. This flag *correlates* the values of an arbitrary number of variables — usually two. If the values of all variables are lists then *--unroll* creates as many rows in the resulting table as elements in the shortest list. For example, to show how the quality of the solutions generated by **arvand** in problem *011* of the domain *openstacks* improved over time:

```
$ /report.py --summary seq-sat.snapshot --quiet
  --planner 'arvand' --level problem --domain 'openstacks' --problem '011'
  --variable timesols values --unroll

 name: report
+---------+------------+---------+----------+--------+
| planner |   domain   | problem | timesols | values |
+---------+------------+---------+----------+--------+
|  arvand | openstacks |   011   |    19    | 126.0  |
|  arvand | openstacks |   011   |    27    | 125.0  |
|  arvand | openstacks |   011   |    63    | 123.0  |
|  arvand | openstacks |   011   |   173    | 122.0  |
|  arvand | openstacks |   011   |   181    | 121.0  |
|  arvand | openstacks |   011   |   257    | 120.0  |
|  arvand | openstacks |   011   |   266    | 118.0  |
|  arvand | openstacks |   011   |   375    | 116.0  |
|  arvand | openstacks |   011   |   425    | 115.0  |
|  arvand | openstacks |   011   |   478    | 113.0  |
|  arvand | openstacks |   011   |   589    | 112.0  |
|  arvand | openstacks |   011   |   755    | 111.0  |
|  arvand | openstacks |   011   |   876    | 110.0  |
|  arvand | openstacks |   011   |   931    | 109.0  |
+---------+------------+---------+----------+--------+
 legend:
   planner [key]
   domain [key]
   problem [key]
   timesols: elapsed time when each solution was generated (in seconds) [raw data]
   values: final values returned by VAL, one per each *valid* solution file [raw data]

 created by IPCrun 1.0 (Revision: 283), Thu Jul 21 14:22:04 2011
```

Should *--unroll* not have been given, the report would have just issued a single line with one list per variable, which is not the desired effect.

Because `report.py` acknowledges a number of output formats with the flag *--style*, *--unroll* is very useful for creating figures. The available styles are `table`, `octave`, `html`, `excel` and `wiki`. The first is used by default. `octave` shows the same information but in the format of GNU Octave which can be read also by gnuplot. `html` and `wiki` are markup languages to show the same data either in html pages or in the wiki format recognized by MoinMoin. Finally, `excel` creates a file named `report.xls` with the result of the query.

The last directive that affects the output is *--name*. It can be used to give the resulting table an arbitrary name.

4.4 score.py

This section describes all the command-line arguments accepted by `score.py`. For a thorough discussion of the command-line arguments see *Command-line arguments*.

This script automatically generates score tables for a selected subset of domains, planners and problems. As in the case of `report.py` (see *report.py*), this script receives either a results tree (as the one depicted in *The results directory*) or a snapshot —as described in *Snapshots*. Let *origin* denote both a result directory and a snapshot or summary —note that `score.py` does not generate any snapshots and that only `report.py` can do it, for more information refer to the directive `--summarize` in *Inspecting data*. The origin shall refer always to a whole track-subtrack. This is, it is not valid to specify either a directory or a snapshot that points to a planner, domain or problem.

The collection of planners, domains and problems to consider can be refined further with regular expresions with `--planner`, `--domain` and `--problem`.

If the directive `--time` is given, then all measurements are relative to the time interval [0, *time*] (where *time* is the value given to `--time` in seconds). If none is specified, then all results are used. This allows drawing conclusions for different time horizons, others than that used in the experimentation —see the usage of the directive `--timeout` in *invokeplanner.py*.

`score.py` acknowledges up to six different metrics. All of them are described if the directive `--metrics` is given:

quality This is the official metric of both the Sixth and Seventh International Planning Competitions. It computes for each task a score which equals $\frac{Q^*}{Q}$ where Q^* is the quality of the best plan found for this particular task and Q stands for the quality of the plan produced by this planner.

solutions It gives one point to every planner that solves the current task and zero otherwise.

time0 Computes the score of a planner for a given task as the quotient $\frac{T^*}{T}$ where T^* is the minimum time required by any planner to solve the same task and T is the time it took this particular planner to solve the same task.

All times below 1 second are considered to be exactly equal to 1 second. In other words, differences below one second are considered to be negligible.

time1 Computes the score of a planner for a given task as the quotient $\frac{1}{1+\log\left(\frac{T}{T^*}\right)}$ where T^* is the minimum time required by any planner to solve the same task and T is the time it took this particular planner to solve the same task.

All times below 1 second are considered to be exactly equal to 1 second. In other words, differences below one second are considered to be negligible.

time2 Computes the score of a planner for a given task as the quotient $\frac{\log(1+T^*)}{\log(1+T)}$ where T^* is the minimum time required by any planner to solve the same task and T is the time it took this particular planner to solve the same task.

qt It computes for each planner and task a tuple (Q, T) where Q stands for the quality of the best solution found by the same planner and T is the time (in seconds) it took for the planner to find it. Next, it gives to each planner a score that equals the number of tuples it pareto-dominates for the same task.

(Q, T) is said to pareto-dominate (Q', T') if and only if $Q \leq Q'$ and $T \leq T'$

All the scores are shown in the form of tables, one per domain that meet the regular expression given to `--domain`. Besides, if more than one domain is given, the script computes a final table called *ranking* with the sum of the scores of all the previous tables. Each table can be given a name with `--name`. This directive accepts placeholders which are symbolized with the dollar sign `$`. In particular, there are five recognized variables: *$track*, *$subtrack*, *$domain*,

$date and *$time* which are substituted with the particular track, subtrack, domain, current date and time. The default value of this variable is *$track-$subtrack: $domain ($date)* [1].

For example, the following command:

```
$ /score.py --summary seq-opt.results.snapshot
            --planner 'f' --domain 'sokoban|parcprinter' --time 10
```

will output the score tables for those planners of the sequential optimization track that start with the letter *f* in the domains *sokoban* and *parcprinter* taking into account only the results that were produced in the first 10 seconds — though the specified snapshot contains the results of running the planners up to 1800 seconds. The metric used is the default one `quality` and the output (discussed below) is just shown as ASCII tables:

```
seq-opt: parcprinter (Mon Dec 19 23:35:01 2011)
+-------+-------------+--------+--------+----------+------------+
|  no.  | fd-autotune | fdss-1 | fdss-2 | forkinit |    best    |
+-------+-------------+--------+--------+----------+------------+
|  000  |     1.00    |  1.00  |  1.00  |   1.00   | 375821.00  |
|  001  |     1.00    |  1.00  |  1.00  |   1.00   | 438047.00  |
|  002  |     1.00    |  1.00  |  1.00  |   1.00   | 510256.00  |
|  003  |     1.00    |  1.00  |  1.00  |   ---    | 876094.00  |
|  004  |     1.00    |  1.00  |  1.00  |   1.00   | 519232.00  |
|  005  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  006  |     1.00    |  ---   |  ---   |   ---    | 1145132.00 |
|  007  |     1.00    |  1.00  |  1.00  |   1.00   | 751642.00  |
|  008  |     1.00    |  1.00  |  1.00  |   1.00   | 693064.00  |
|  009  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  010  |     1.00    |  ---   |  ---   |   ---    | 1216462.00 |
|  011  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  012  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  013  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  014  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  015  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  016  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  017  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  018  |     ---     |  ---   |  ---   |   ---    |    ---     |
|  019  |     1.00    |  ---   |  ---   |   ---    | 1270874.00 |
| total |    10.00    |  7.00  |  7.00  |   6.00   |            |
+-------+-------------+--------+--------+----------+------------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.2 (Revision: 295), Mon Dec 19 23:35:01 2011


seq-opt: sokoban (Mon Dec 19 23:35:01 2011)
+-------+-------------+--------+--------+----------+-------+
|  no.  | fd-autotune | fdss-1 | fdss-2 | forkinit |  best |
+-------+-------------+--------+--------+----------+-------+
|  000  |     1.00    |  1.00  |  1.00  |   1.00   |  9.00 |
|  001  |     1.00    |  1.00  |  1.00  |   1.00   | 37.00 |
|  002  |     1.00    |  1.00  |  1.00  |   1.00   | 29.00 |
|  003  |     1.00    |  1.00  |  1.00  |   1.00   | 29.00 |
|  004  |     ---     |  1.00  |  1.00  |   ---    | 50.00 |
|  005  |     ---     |  1.00  |  1.00  |   ---    | 35.00 |
|  006  |     1.00    |  1.00  |  1.00  |   1.00   | 30.00 |
```

[1] When using the dollar sign `$` in the command line, the shell will always try to expand it to the values of environment variables. This is known as *interpolation*. To avoid it, strings containing dollar signs shall be embraced between single quotes as in '$track-$subtrack.$planner-build'

```
|  007  |     1.00    |  1.00  |  1.00  |   1.00   | 19.00 |
|  008  |     1.00    |  1.00  |  1.00  |   1.00   | 15.00 |
|  009  |     1.00    |  1.00  |  1.00  |   1.00   |  8.00 |
|  010  |     1.00    |  1.00  |  1.00  |   1.00   | 20.00 |
|  011  |     1.00    |  1.00  |  1.00  |   1.00   |  2.00 |
|  012  |     ---     |  ---   |  ---   |   ---    |  ---  |
|  013  |     1.00    |  ---   |  ---   |   1.00   | 32.00 |
|  014  |     ---     |  ---   |  ---   |   ---    |  ---  |
|  015  |     ---     |  ---   |  ---   |   ---    |  ---  |
|  016  |     ---     |  ---   |  ---   |   ---    |  ---  |
|  017  |     1.00    |  1.00  |  1.00  |   ---    | 10.00 |
|  018  |     ---     |  ---   |  ---   |   ---    |  ---  |
|  019  |     ---     |  ---   |  ---   |   ---    |  ---  |
| total |    12.00    | 13.00  | 13.00  |  11.00   |       |
+-------+-------------+--------+--------+----------+-------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.2 (Revision: 295), Mon Dec 19 23:35:01 2011


seq-opt: ranking (Mon Dec 19 23:35:01 2011)
+-------------+---------+-------------+-------+
|   planner   | sokoban | parcprinter | total |
+-------------+---------+-------------+-------+
| fd-autotune |  12.00  |    10.00    | 22.00 |
|    fdss-1   |  13.00  |     7.00    | 20.00 |
|    fdss-2   |  13.00  |     7.00    | 20.00 |
|   forkinit  |  11.00  |     6.00    | 17.00 |
|    total    |  49.00  |    30.00    |       |
+-------------+---------+-------------+-------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.2 (Revision: 295), Mon Dec 19 23:35:01 2011
```

Finally, `score.py` can produce the output in a variety of formats. It recognizes at least the same ones described in *Inspecting data* and, additionally, `latex`. If used, it creates a LaTeX file called `matrix.tex`. Each page is divided in two halves: the upper contains the table whereas the lower half shows up a matrix of color codes with the following meanings:

Red boxes invalid entry. The planner generated a solution but it was considered invalid by the Automatic Validation Tool VAL

Yellow boxes empty solution. The planner never found a solution for this task.

Gray boxes solved tasks. It uses gray levels to mean scores. The darker the better.

Since the resulting LaTeX file uses ps-tricks it cannot be processed directly with **pdflatex**. Instead, a makefile is given in the same directory where this package resides. To produce the corresponding pdf file just type:

```
$ make filename.pdf
```

where `filename` stands for the name of the LaTeX file —`matrix` in this case.

4.5 tscore.py

This script behaves much the same like `score.py` but with a key difference. While it acknowledges the same directives than the previous script (though no LaTeX output is supported), it just computes how the score of all planners evolve over time on all the selected domains.

These figures are computed taking for each planner the time when they generate a solution. This is, at each time instant, where at least one planner among those selected by the regular expression given in *`--planner`* found a solution to at least one problem in a particular domain, the score of all planners is computed. This process produces a curve that shows how the score of each planner evolved over time at precise time instants.

Since this computation can be costly (a matter of minutes in the larger tracks if direct access to the result directories is being performed instead of snapshots), the script also acknowledges a new flag *`--labels`*. This directive allows the user to specify a number of time points which are drawn from the original list of time instants at regular intervals —this implies that the final number of points drawn might not be exactly equal to the number requested by the user, though it will be always as close as possible.

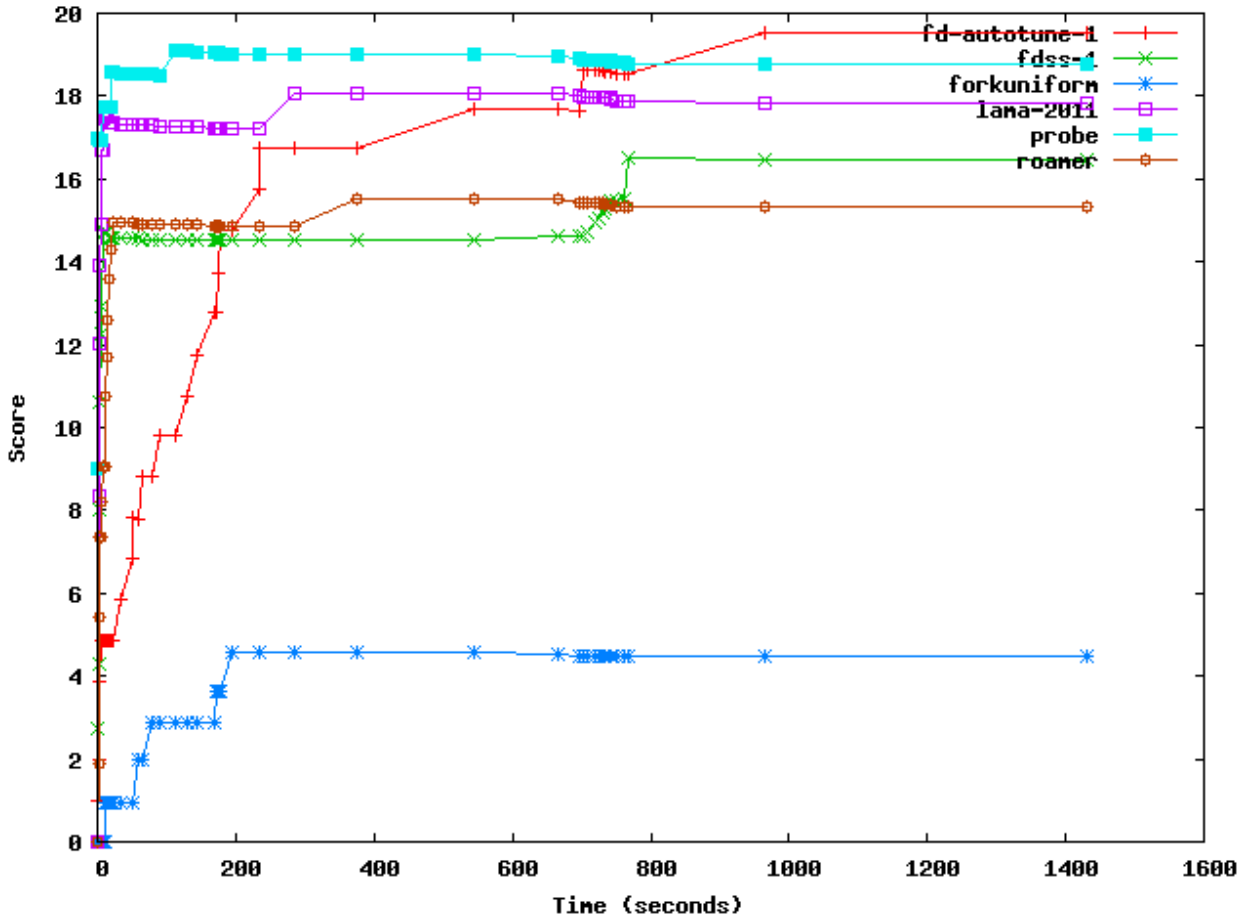
For example, the following command computes how the score evolves over time for the six top ranked planners in the Seventh International Planning Competition (if more than one variant of the same planner ranked among these, the best is picked up) for the domain *barman*:

```
$ ./tscore.py --summary seq-sat.snapshot --metric quality
  --planner "lama-2011|fdss-1|fd-autotune-1|roamer|forkuniform|probe" --domain "barman"
  --style octave --quiet > salida.m
```

Now, if all matrices in the output file *`salida.m`* are removed but *scores_barman*, the following commands in gnuplot:

```
gnuplot> set xlabel "Time (seconds)"
gnuplot> set ylabel "Score"
gnuplot> set terminal png
gnuplot> set output "barman.png"
gnuplot> plot "salida.m" using 1:2 with linesp title "fd-autotune-1",
              "salida.m" using 1:3 with linesp title "fdss-1",
              "salida.m" using 1:4 with linesp title "forkuniform",
              "salida.m" using 1:5 with linesp title "lama-2011",
              "salida.m" using 1:6 with linesp title "probe",
              "salida.m" using 1:7 with linesp title "roamer"
```

produce the following output:



which can be used to draw a number of interesting conclusions.

Even if only one domain meets the regular expression specified with `--domain` this scripts shows up an overall ranking table at the end. This table takes the time instant where at least one planner solved one task in any of the domains specified. Besides, while the tables generated per domain list planners in alphabetical order, the overall ranking table shows them in decreasing order of total score.

4.6 test.py

In most cases, looking at the number of problems solved, their plan quality or other characteristics is not enough to judge whether one planner performs better than another. This problem is rather typical in many fields of Science (including Artificial Intelligence) and the usual approach consists of performing statistical tests. Since the module `IPCReport` already provides a facility to access data (see *report.py*), it is almost straightforward to provide another script to perform statistical tests over the same data. This is the target of `test.py`. This script uses `report.py` transparently to the user to retrieve data from a snapshot or summary (see *Snapshots*) or results tree directory (see *The results directory*) and perform the indicated statistical tests over the resulting series.

The script `test.py` implements four different statistical tests. Since parametrical statistical tests make questionnable assumptions about the distribution of data, and besides most series are more likely to be relatively short (e.g., in the seventh International Planning Competition there were 20 planning tasks per domain so that most series have *n=20* samples which is regarded in some texts as being borderline between a small and large set) three of them are nonparametric. However, because of its popularity, a fourth one which is parametric is included as well:

Mann-Whitney U-test It compares two samples that are independent, or not related. It assesses whether one of two samples of independent observations tends to have larger values than the other. The test automatically corrects for ties and by default uses a continuity correction. The reported *p*-value is for a one-tailed hypothesis, i.~e., when information about whether one sample have larger values than the other is provided. To get the two-tailed *p*-value (i.~e., when the null hypothesis is rejected if the test statistic is either too small or too large) the returned *p*-value has to be multiplied by two.

Wilcoxon signed rank test In contraposition to the previous test, the Wilcoxon signed rank test is a two-tailed nonparametric statistical procedure for comparing two samples that are paired, or related. It tests the null hypothesis that both samples come from the same distribution.

This test has been extensively used in the analysis of previous International Planning Competitions, mostly in the third and fifth.

Binomial test It is an exact test used with dichotomous data —that is, when each individual in the sample is classified in one of two categories such as success/failure. It provides statistical significance of deviations from a binomial distribution with *p=0.5*.

The use of this test in the context of Automated Planning was originally proposed by Hoffmann and Nebel to provide statistical significance of the differences in performance of their planner **FF** when using different combinations of enhancements

t-Test It is the parametric equivalent test of the Wilcoxon signed rank test. This is a two-tailed test for the null hypothesis that two independent samples have identical average (expected) values

One restriction of all of these tests, however, is that they just compare two series of data. Other tests such as, the Kolmogorov-Smirnov one-sample test to determine if a data sample meets acceptable levels of normality or the Friedmann or the Kruskal-Wallis *H*-tests to compare three or more samples (either related or unrelated respectively), are not currently implemented. Instead, all these statistical tests perform pairwise comparisons of an arbitrary number of series and provide the *p*-value of each pair according to the selected statistical procedure. If the resulting *p*-value is less or equal than the critical value that corresponds to a particular level of risk α, the null hypothesis is rejected and the alternate or research hypothesis is accepted instead. Typical values of the level of risk are α*=0.05, 0.01* and *0.001* which stand for a probability of 95%, 99% and 99.9% respectively that any observed statistical difference will be real and not due to chance.

This script retrieves data from `report.py` (see *report.py*) transparently to the user so that it acknowledges the same directives that can be used with exactly the same purpose but just one restriction: only one variable (with `--variable`) can be provided so that only single-valued series are allowed. Once `report.py` has been silently invoked it retrieves a unique table of data which is split in as many series as primary keys are present in the table by `test.py`. For example, the following query returns the number of problems *apparently* solved (i.e., that the planner claims that it solved) and those that are successfully solved (i.e., validated with VAL) in the woodworking domain by planners **fdss-2**, **lmcut** and **gamer**:

```
$ ./report.py --summary ./seq-opt.results.snapshot --variable solved oksolved
              --domain woodworking --planner 'fdss-2|lmcut|gamer'
```

```
+---------+-------------+---------+--------+----------+
| planner |    domain   | problem | solved | oksolved |
+---------+-------------+---------+--------+----------+
|  fdss-2 | woodworking |   000   |  True  |   True   |
|  fdss-2 | woodworking |   001   |  True  |   True   |
|  fdss-2 | woodworking |   002   |  True  |   True   |
|  fdss-2 | woodworking |   003   |  True  |   True   |
|  fdss-2 | woodworking |   004   |  True  |   True   |
|  fdss-2 | woodworking |   005   |  True  |   True   |
|  fdss-2 | woodworking |   006   |  True  |   True   |
```

```
|  fdss-2 | woodworking |   007   |  True  |   True   |
|  fdss-2 | woodworking |   008   |  True  |   True   |
|  fdss-2 | woodworking |   009   |  True  |   True   |
|  fdss-2 | woodworking |   010   | False  |  False   |
|  fdss-2 | woodworking |   011   | False  |  False   |
|  fdss-2 | woodworking |   012   | False  |  False   |
|  fdss-2 | woodworking |   013   | False  |  False   |
|  fdss-2 | woodworking |   014   |  True  |   True   |
|  fdss-2 | woodworking |   015   | False  |  False   |
|  fdss-2 | woodworking |   016   | False  |  False   |
|  fdss-2 | woodworking |   017   | False  |  False   |
|  fdss-2 | woodworking |   018   | False  |  False   |
|  fdss-2 | woodworking |   019   | False  |  False   |
|  gamer  | woodworking |   000   |  True  |   True   |
|  gamer  | woodworking |   001   |  True  |   True   |
|  gamer  | woodworking |   002   |  True  |   True   |
|  gamer  | woodworking |   003   |  True  |   True   |
|  gamer  | woodworking |   004   |  True  |   True   |
|  gamer  | woodworking |   005   |  True  |   True   |
|  gamer  | woodworking |   006   |  True  |   True   |
|  gamer  | woodworking |   007   |  True  |   True   |
|  gamer  | woodworking |   008   |  True  |   True   |
|  gamer  | woodworking |   009   |  True  |   True   |
|  gamer  | woodworking |   010   |  True  |   True   |
|  gamer  | woodworking |   011   |  True  |   True   |
|  gamer  | woodworking |   012   |  True  |   True   |
|  gamer  | woodworking |   013   |  True  |   True   |
|  gamer  | woodworking |   014   |  True  |   True   |
|  gamer  | woodworking |   015   |  True  |   True   |
|  gamer  | woodworking |   016   | False  |  False   |
|  gamer  | woodworking |   017   | False  |  False   |
|  gamer  | woodworking |   018   |  True  |   True   |
|  gamer  | woodworking |   019   | False  |  False   |
|  lmcut  | woodworking |   000   |  True  |   True   |
|  lmcut  | woodworking |   001   |  True  |   True   |
|  lmcut  | woodworking |   002   |  True  |   True   |
|  lmcut  | woodworking |   003   |  True  |   True   |
|  lmcut  | woodworking |   004   |  True  |   True   |
|  lmcut  | woodworking |   005   |  True  |   True   |
|  lmcut  | woodworking |   006   |  True  |   True   |
|  lmcut  | woodworking |   007   |  True  |   True   |
|  lmcut  | woodworking |   008   |  True  |   True   |
|  lmcut  | woodworking |   009   |  True  |   True   |
|  lmcut  | woodworking |   010   | False  |  False   |
|  lmcut  | woodworking |   011   | False  |  False   |
|  lmcut  | woodworking |   012   | False  |  False   |
|  lmcut  | woodworking |   013   | False  |  False   |
|  lmcut  | woodworking |   014   |  True  |   True   |
|  lmcut  | woodworking |   015   |  True  |   True   |
|  lmcut  | woodworking |   016   | False  |  False   |
|  lmcut  | woodworking |   017   | False  |  False   |
|  lmcut  | woodworking |   018   | False  |  False   |
|  lmcut  | woodworking |   019   | False  |  False   |
+---------+-------------+---------+--------+----------+
```

The primary key in this case is **planner** which is instatiated to **fdss-2**, **gamer** and **lmcut**. Therefore, `test.py` automatically creates three series with the values of a single variable for these keys —in the previous example **solved** was shown also to exemplify below how two variables can be used simultaneously with the directive *--filter*.

Observing the number of solved problems in this particular domain, it turns out that **gamer** seems to be the best (solving 17 problems) followed by **lmcut** (which solves 12) and **fdss-2** which solves 11. However, performing a statistical test over these series will provide a more reliable impression of the relative performance of these planners. Since the data in these series are dichotomic (it only takes the values `True` and `False`) a *Binomial test* is performed to know whether one planner performs better than another. Initially, one can directly ask `test.py` to perform the statistical test just passing by the same parameters but with just one single variable of interest, **oksolved** along with the particular selection of the statistical test to perform with the option *--test*:

```
$ ./test.py --summary ./seq-opt.results.snapshot --variable oksolved
            --domain woodworking --planner 'fdss-2|lmcut|gamer' --test bt

   Revision
   Date

   ./test.py 1.3

   ------------------------------------------------------------------------------
   * snapshot       : ./seq-opt.results.snapshot
   * tests          : ['Binomial test']
   * name           : report
   * level          : None
   * planner        : fdss-2|lmcut|gamer
   * domain         : woodworking
   * problem        : .*
   * variable       : ['oksolved']
   * filter         : None
   * matcher        : all
   * noentry        : -1
   * unroll         : False
   * sorting        : []
   * style          : table
   ------------------------------------------------------------------------------


   name: report
  +--------+----------+-------+---------+
  |        |  fdss-2  | gamer |  lmcut  |
  +--------+----------+-------+---------+
  | fdss-2 |   ---    |  1.0  |   1.0   |
  | gamer  | 0.015625 |  ---  | 0.03125 |
  | lmcut  |   0.5    |  1.0  |   ---   |
  +--------+----------+-------+---------+
   Binomial test : Perform a binomial two-sided sign test. It computes the number n of times that the
   shown in the row behaves differently than the serie shown in the column. It returns the probabilit
   according to a binomial distribution with p=0.5 that the number of times that the serie shown in t
   takes values larger than the serie shown in the column equals at least the number of times that th
   difference was observed.
   If this probability is less or equal than a given threshold, e.g., 0.01, 0.05 or 0.1, then reject
   the null hypothesis and assume that the serie shown in the column is significantly smaller

   created by IPCtest 1.3 (Revision: 312), Sun Jul 15 17:21:38 2012
```

It is possible to invoke `test.py` with the option *--tests* to get a full list of all the implemented statistical tests along with a description of their use and purpose. It is also feasible to request an arbitrary number of statistical tests passing them altogether after *--test* —as in *--test wx mw* which requests simultaneously the Wilcoxon Signed-rank test and Mann Whitney U test.

From the preceding figure, it seems that **fdss-2** and **lmcut** perform better than **gamer** with a confidence level α=*0.05* (i.e., with a probability equal to 95%). This result goes against the original intuition. The reason is that the Binomial

test tests whether the planner in the colum has values smallers than the planner shown in the row. Since `False` is considered to be smaller than `True` in most computing languages (including Python) the results are clearly misleading. To correct the results it is neccessary to provide a larger value to those problems that were not solved.

Hence, the first step consists of *filtering* data. In this case, the variable of interest is **solved** (whether the planner provided at least one solution to a single planning task) which is filtered by **oksolved** —whether the plan found is valid or not. A filter (**oksolved** in the following example) sets the value of a particular sample to the constant `NOENTRY` if it is `False` and passes the value of the selected variable (**solved**) in case it is `True`. In other words, it filters the input data according to a secondary variable. The primary variable is selected with *`--variable`* (as in `report.py`), whereas the secondary variable is selected with *`--filter`*. However, filtering data poses a new question: *When comparing two series, what to do with those entries in one serie whose value equals NOENTRY?*

Sometimes it is desirable to compare only those entries from two series where both elements have been filtered —these are known as **double hits**. In other cases, it might be better to preserve those entries where only one serie has the value `NOENTRY`. The third alternative consists of comparing both series even if the same entry contains `NOENTRY` for both series. This selection can be performed with *`--matcher`* which accepts the values `and`, `or` and `all` to match two series as indicated before respectively:

and It only accepts those entries where both series have values different than `NOENTRY`

or It rejects only those entries where both series have values equal to `NOENTRY`

all It accepts all entries processing both series in their current format

Finally, it is safe to set the value of all entries equal to `NOENTRY` to a particular value which depends upon the *Null Hypothesis* used. In the running example, it is that the distribution of problems solved by one planner is the same than the problems solved by a different planner. To force the statistical test to consider those problems unsolved (or which are not valid) as being worse than those that have been solved it is a must to set `NOENTRY` (which would correspond to either problems unsolved or solved problems which are not valid) to a large value. This is done with the option *`--noentry`*.

In the following example, the same test shown above is performed again but this time: first, the values of the variable **solved** are filtered with the variable **oksolved** to make sure that they are valid plans with the option *`--filter oksolved`*; second, all entries where both series have the value `NOENTRY` are discarded with *`--matcher or`*; thirdly, all the resulting entries with the value `NOENTRY` are set to 100 to force the statistical test to consider them worse (as they are larger values than `True` which just equals the integer 1) than those entries that correspond to valid solutions:

```
$ ./test.py --summary ./seq-opt.results.snapshot --variable solved --filter oksolved
            --matcher or --noentry 100 --domain woodworking
            --planner 'fdss-2|lmcut|gamer' --test bt

   Revision
   Date

   ./test.py 1.3

   --------------------------------------------------------------------------------
   * snapshot       : ./seq-opt.results.snapshot
   * tests          : ['Binomial test']
   * name           : report
   * level          : None
   * planner        : fdss-2|lmcut|gamer
   * domain         : woodworking
   * problem        : .*
   * variable       : ['solved']
   * filter         : ['oksolved']
   * matcher        : or
   * noentry        : 100
```

```
 * unroll         : False
 * sorting        : []
 * style          : table
 ------------------------------------------------------------------------------


 name: report
+--------+--------+----------+-------+
|        | fdss-2 |  gamer   | lmcut |
+--------+--------+----------+-------+
| fdss-2 |  ---   | 0.015625 |  0.5  |
| gamer  |  1.0   |   ---    |  1.0  |
| lmcut  |  1.0   | 0.03125  |  ---  |
+--------+--------+----------+-------+
 Binomial test : Perform a binomial two-sided sign test. It computes the number n of times that the
 shown in the row behaves differently than the serie shown in the column. It returns the probabilit
 according to a binomial distribution with p=0.5 that the number of times that the serie shown in t
 takes values larger than the serie shown in the column equals at least the number of times that th
 difference was observed.
 If this probability is less or equal than a given threshold, e.g., 0.01, 0.05 or 0.1, then reject
 the null hypothesis and assume that the serie shown in the column is significantly smaller

 created by IPCtest 1.3 (Revision: 312), Sun Jul 15 21:00:19 2012
```

As it can be seen, the results indicate now that **gamer** performs better (i.e., it has values smaller) than **lmcut** with a confidence level larger than 96% and **fdss-2** with a confidence level larger than 98%. It seems that the *Research Hypothesis* that **gamer** outperforms the other two planners can be accepted only with the most conservative of the typical confidence levels, 95% —since the *p*-values retrieved are only smaller than 0.05 but not than other typical values, 0.01 and 0.001.

Finally, this script acknowledges all the different styles provided by `report.py` with the directive `--style` so that the same tables can be shown in the markup languages html and wiki, octave files and also in excel worksheets. New in version 1.2: Command-line arguments parsed with argparseNew in version 1.1: Welcome `IPCTools`!

CHAPTER

FIVE

IPCTOOLS

This package was developed after the IPC to ease the data analysis, in particular across tracks/subtracks so that they are available only from version 1.1. As a matter of fact, the modules contained in it serve to create a new (artificial) track with the results of different planners from different tracks with respect to the same set of problems. As soon as data is arranged as explained in *The results directory* it is then feasible to run the reporting tools described in *IPCReport*

It consists of two different modules: `copy.py` and `rename.py`

5.1 Command-line arguments

As a general rule all of these programs have, at least, the following two flags:

-h, --help	provide a brief description of the main purpose of the script and presents all the available flags
-V, --version	shows the current version of the script along with the latest svn release that affected it

The package is located in:

```
svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral/IPCTools
```

All these modules have been developed with Python 2.x

An example of the usage of these package is shown in *Third practical case*.

5.2 copy.py

The first step to perform analysis across tracks consists of creating an artificial new track. For example, to compare the performance of the winners of the sequential satisficing track and the sequential optimal track (**lama-2011** and **fdss-1** respectively) with regard to the same set of planning tasks, say those in the sequential optimal track, a new track shall be created. In this example it will be known as `seqopt-opt`. The first contents of this track are created by copying the results of one of the planners to be considered in comparison. `copy.py` serves explicitly to this goal.

Since `copy.py` copies files beneath a target directory, it is a good idea to create it explicitly:

```
$ mkdir ~/tmp/seqopt-opt
```

This comand will create a new directory to host the new track in the temporary directory. Usually, it is a good idea to start afresh with a new directory since the source directories shall not exist in the destination directory: creating a directory from scracth clearly guarantees this.

This module accepts two directives: *--source* and *--destination*. While the latter has to be unique, the former accepts wildcards and can be specified an arbitrary number of times. The module copies all directories matching the source expressions into the destination directory.

In our running example, it is assumed that the results of running **lama-2011** with the problems specified in the sequential satisficing track are available in a directory `~/tmp/results/sat-opt/lama-2011`. Hence, the following command replicates its contents but this time under a different track/subtrack:

```
$ copy.py --source ~/tmp/results/sat-opt/lama-2011
          --destination ~/tmp/seqopt-opt
```

Two important notes:

1. The script automatically realizes that the source directory refers to a planner. Alternatively, it is possible to specify domains or problems.

2. As discussed in *The results directory* the new results directory shall be created according to the name of the planners/domains/tasks stored therein. Therefore, `copy.py` renames accordingly all its subdirectories as if it would have been originally created by `invokeplanner.py` —see *invokeplanner.py*.

As mentioned above, other directories can be copied to the same destination directory either by providing additional *--source* directives or just running the script again. For example, executing now (assuming that the results of **fdss-1** are stored in the directory specified below):

```
$ copy.py --source ~/tmp/results/seq-opt/fdss-1
          --destination ~/tmp/seqopt-opt
```

the new track `seqopt-opt` will be created with the results of two planners: **lama-2011** and **fdss-1**. Its structure resembles the organization of a legal results directory so that it can be safely examined by the reporting tools —see *IPCReport*

5.3 rename.py

While it might happen that some people might change their mind and want to rename a planner/domain/problem (as it actually happened while running the Seventh International Planning Competition), most people will not use this module directly. It is intended, as its name suggests, to rename a directory which contains a planner, a domain or a particular problem in a results directory, affecting the names of nested files if necessary.

It accepts only two directives: *--directory* and *--name*. While *--directory* has to specify the whole path to the directory to rename (either in absolute or relative formats), the value of the flag *--name* shall be only the new name of the last folder specified with *--directory*.

For example, to rename the planner **Multiplan.q** that took part in the learning track of the Seventh International Planning Competition to **PbP2.q**, it suffices with:

```
$ rename.py --directory ~/tmp/results/lrn-sat/Multiplan.q
            --name PbP2.q
```

This module is automatically invoked by `copy.py` when copying planners/domains/problems across different tracks so that the resulting tree structure is properly named.

CHAPTER

SIX

IPCPRIVATE

This package contains code that is privative of the Seventh International Planning Competition. While it might be useful to reproduce the official results of the Competition, it is not intended for a general use. Hence, the suffix *Private*.

As a matter of fact, this package consists of a single module `unsharaabi.py` which is described in the next section.

6.1 Unsharaabi.py

The name of this module comes from the fact that it explicitly undo most of the work of the planner **Sharaabi** in the plan solution files. This planner internally invokes the service of another one (**SAPA**) which writes a lot of ancilliary data (in the order of tens of Megabytes) to its standard output. This, being captured by **Sharaabi**, is sent to the `plan.soln` files. While writing a lot of data is not necessarily a problem, this extra information makes it impossible for the validation tool, VAL, to parse the solutions.

Because the original author of this planner did not find a way to avoid writing this extra amount of information to the plan solution files, a script is provided here to process these files. The package is located in:

`svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral/IPCPrivate`

The script acknowledges (other than the usual flags *--help* and **–version**) the directive *--directory* which specifies the root directory of the tree structure to process.

Note: All plan solution files either in the specified directory with *--directory* or under it are processed. The script does not check the name of the planner who produced every plan solution file. If it is invoked on the directory of another planner, the results are unpredictable.

Upon completion, the script leaves a copy of each `plan.soln` file generated so far with its name reversed. The new file contains only the text that was embraced between *Original plan returned by Sapa* and *End original plan* in the original plan solution file. If no text was found between these lines or even if these lines did not appear, an empty file results which is then deleted to signal that no solution was found.

CHAPTER

SEVEN

PRACTICAL CASES

In this chapter, a number of practical cases is shown to exemplify most of the concepts explained in the previous chapters. Links to the relevant parts of this manual are provided when applicable for the interested reader. Bear in mind that this tutorial does not cover all the possibilities offered by the software developed at the Seventh International Competition. Therefore, referring to the manual pages is a good idea to get a detailed picture of all the possibilities.

In the first example, the temporal satisficing planners **yahsp2** and its multithreaded version **yahsp2-mt** are evaluated in two domains: crewplanning and turnandopen. None of these planners is known to have any particular requirement —however, it will be shown that they can be easily identified and met when following the procedure depicted below.

The second example shows how to set-up the environment for running experiments with the software developed at the Seventh International Planning Competition. This example has been explicitly created to encourage users to use this software for evaluating their own planners —though official entries of the Seventh International Planning Competition have been chosen to allow reproducibility. In particular, the planner **probe** and the domain openstacks are used here.

The third example shows how to compare the performance of a particular planner with the performance of other planners that entered the Seventh International Planning Competition. Here, the planner **probe** will be compared in the opestacks domain to the performance of the winner of the Sequential Satisficing (seq-sat) track **lama-2011**.

The fourth example discusses how to perform a statistical test to derive accurate figures about the relative performance of two planners on the same problem. In this case, the planners **yahsp2** and its multithreaded version **yahsp2-mt** are used with problem 001 of the domain openstacks.

Readers are strongly encouraged to follow the examples in the same order they are given here: while the main goal of the first practical case is to show how to run particular experiments with planners and domains stored in a particular svn server and how to inspect the resulting data, the second and third practical cases are intended to show how to set-up their own svn servers and perform comparisons with the performance of entrants of the Seventh International Planning Competition —and, in general, to any other planner.

7.1 First practical case

In this example, the temporal satisficing planners **yahsp2** and its multithreaded version **yahsp2-mt** are evaluated in two domains: crewplanning and turnandopen. None of these planners is known to have any particular requirement —however, it will be shown that they can be easily identified and met when following the procedure depicted here.

The time limit will be set to 10 seconds so that this tutorial can be run without waiting too long for the results. Besides, the memory limit will be set to 2 Gb which is rather typical in most modern computer configurations.

It is assumed that this exercise is performed in the same machine where the results are about to be stored and examined. Nevertheless, nothing prevents experienced users to run the experiments in remote computers using ssh commands provided that they retrieve the result directories as explained in *The results directory*. However, this topic is not covered here.

The tutorial is divided in the following parts:

Setting up A linux computer is configured to meet all the dependencies of the IPC software; next, the software is installed in a single command and the INI configuration file is created

Running The competition is run for the abovementioned planners and domains. The time limit is 10 seconds and the memory bound is set to 2 Gb —which is assumed to be a minimum in modern computer configurations.

Reporting The results are examined. First a snapshot is created and then various reports are created to show how different values can be retrieved in a wide variety of formats. Finally, the score sheet used at the International Planning Competition is used to rank planners.

In those cases where the output is intentionally ommitted an ellipsis is inserted instead.

7.1.1 Setting up

The first step consists of configuring a computer to run the experiments and to examine the results. While it is not necessary at all that both tasks are performed in the same computer, this is not discussed here since it goes beyond the scope of this document. However, experienced users will have no trouble in getting this done. As a matter of fact, that was actually the layout during the Seventh International Planning Competition.

It is assumed as well, that users have privileges to install software. If not, contact your system administrator or install them in your account.

In this example, GNU/Linux Ubuntu 11.04 has been installed in a computer —as a matter of fact, the experiments were run on a VMWare Virtual Machine with 2Gb of RAM. Of course, python 2.7 and svn have to be available in the computer as well as the software required by the selected planners to be built. While Python 2.7 is available by default in the latest version of GNU/Linux at the time of writing this manual, this does not apply to subversion. Therefore, to download and install svn:

```
$ sudo apt-get install subversion
```

In any case, before moving on make sure that both Python 2.7 and subversion are available in your computer:

```
$ python --version
$ svn --version
```

If they are, the previous commands shall report the current versions installed in your machine.

Since we want to use the software developed at the Seventh International Planning Competition in its full glory, all the dependencies of all modules shall be met before moving on:

- The package `IPCData` have dependencies with the Python packages **ConfigObj** and **Subversion Python** as explained in *Dependencies*
- The package `IPCReport` has a single dependency with the Python package **pyExcelerator** as discussed in *Dependencies*
- Finally, both packages use the PrettyTable module which is distributed separately

As soon as they are all available, the software of the Seventh International Planning Competition can be checked out from the official repository (or any other one of your choice provided that you have a private copy) and configured accordingly before running the experiments. All of these steps are described in detail in the following subsections.

ConfigObj

In its authors' words: ConfigObj *is a simple but powerful config file reader and writer: an ini file round tripper. Its main feature is that it is very easy to use, with a straightforward programmer's interface and a simple syntax for config*

files [1].

The tarball `configobj.zip` is downloaded from http://www.voidspace.org.uk/python/configobj.html and unzipped in a temporary directory [2]. Next, the installation process is invoked with the script `setup.py`:

```
$ unzip configobj.zip
  ...
$ cd configobj-4.7.2
$ sudo python ./setup.py install
```

Subversion Python

According to the authors: *The* pysvn *project's goal is to enable tools to be written in Python that use Subversion* [3], such as the `IPCData` package developed for the Seventh International Planning Competition.

This package is distributed also as an Ubuntu package. Therefore, the installation process consists simply of executing:

```
$ sudo apt-get install python-svn
```

pyExcelerator

The pyExcelerator Python package allows the creation of Excel worksheets with various features including decorators, splitters, etc.

As it turned out with ConfigObj, it is necessary to download the tarball and to install it. The whole process is performed in a temporary directory:

```
$ unzip pyexcelerator-0.6.4.1.zip
  ...
$ cd pyexcelerator-0.6.4.1
$ sudo python ./setup.py install
```

And pyExcelerator is available in the computer [4].

NumPy and SciPy

The installation instructions of NumPy and SciPy (that cover various platforms) provide a step by step procedure to install the software form the source code. However, in a Linux box it is easier just to install the apt packages as follows:

```
$ sudo apt-get install python-numpy
  ...
$ sudo apt-get install python-scipy
  ...
```

prettyTable

In its authors' words: PrettyTable *is a simple Python library designed to make it quick and easy to represent tabular data in visually appealing ASCII tables*.

[1] While Python also provides a wrapper for accessing INI configuration files, this was found to be very complete and easy to deal with.

[2] The current version of ConfigObj at the time of writing this tutorial was 4.7.2. Obviously, the directory to step in might change if new versions have been deployed later.

[3] Also, the so-called **svn-workbench** is distributed from the same homepage (and it is also available as an Ubuntu package). However, it is not needed at all for using the IPC software.

[4] The current version of pyExcelerator at the time of writing this tutorial was 0.6.4.1. Obviously, the directory to step in might change if new versions have been deployed later.

To use this software it suffices going to the Donwload area in the project homepage and to get the file `PrettyTable.py`. Nevertheless, this is not necessary at all since it is automatically included in the packages developed at the Seventh International Planning Competition.

Downloading the IPC Software

The software of the IPC can be easily retrieved from the official svn server of the Seventh International Planning Competition. Make sure, first, to move to an empty ancilliary directory:

```
$ cd ~/tmp
$ mkdir first-case
$ cd first-case
$ svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/scripts/pycentral
```

Upon completion, all modules *IPCData*, *IPCReport*, *IPCPrivate* and *IPCTools* are checked out and located in the directory `pycentral` that is beneath the current directory.

Configuring the IPC Software

As a matter of fact, there is no need to configure the IPC software unless automatic e-mail notification is desired —for a thorough discussion on the topic visit *seed.py*. However, even if this feature is not going to be used, it might be a good idea to start by running the script `seed.py` since it automatically browses the contents of the svn server to be used and issues a number of warnings if some track/subtracks are found empty. In this example, the svn server chosen is:

```
svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
```

but it could be an arbitrary one [5].

Since this script is located in the package `IPCData` we move there to execute it:

```
$ cd pycentral/IPCData
$ ./seed.py --file ~/.ipc.ini --name "A practical case" --part "Deterministic Part"
            --bookmark svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data
            --wiki http://www.plg.inf.uc3m.es/ipc2011-deterministic
            --cluster pleiades.plg.inf.uc3m.es
            --mailuser ipc2011.pleaides@gmail.com --mailpwd ys98NfyMenRs
            --smtp smtp.gmail.com --port 587
```

All the parameters are mandatory though most of them are almost never used. However, *--mailuser*, *--mailpwd*, *smtp* and *port* are mandatory if the automatic notification by e-mail is desired when running any of the following scripts: `invokeplanner.py`, `builddomain.py`, `buildplanner.py` or `validate.py`. Besides, if the INI configuration file is created there is no need to specify the svn bookmark anymore in any command that requires it since doing it just overrides the default selection specified in the INI configuration file. By default, all scripts seek the INI configuration file in `~/.ipc.ini`.

Finally, the INI configuration file just created contains a detailed explanation of the contents of the svn server which are self-explanatory.

7.1.2 Running

All the previous steps take less than 10 minutes in a computer with a standard connection to Internet. Now, the environment is set up for starting any execution. In what follows, all scripts but `check.py` are exemplified —this

[5] Most readers can find this valuable. An image of the official svn server can be requested by e-mail to carlos.linares@uc3m.es. Then, its contents can be recreated in a private svn server where a number of experiments can be conducted without letting others notice.

script is only suitable for arranging an official competition and serve to very particular goals, see *check.py*.

First, the competition will be run automatically but before that, it is shown how the domains and planners are checked out and compiled. Next, it will be shown how to validate the results.

In general, as it will be shown, the main driver behind the design of the IPC software is to hide the contents and design of the svn server to the user. Only people interested in adding planners and/or domains shall be involved with the contents of the svn server.

Building domains

Users interested in just downloading a number of domains to their computer will find *builddomain.py* handy. In our running example the temporal satisficing domains crewplanning and turnandopen have been selected. To download them:

```
 $ ./builddomain.py --track tempo --subtrack sat
                    --domain crewplanning turnandopen --directory ~/tmp

[Info: using the svn bookmark specified in the ipc ini configuration file]
[      svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data]


[Info: the following domains have been built]
[      [u'crewplanning', u'turnandopen']]
```

First, note that the svn bookmark has not been specified. Indeed, an info message informs that the default selection found in `~/.ipc.ini` is about to be used. Finally, another info message informs that the desired domains have been properly built. As specified with the directive *--directory*, both domains have been built under `~/tmp` —if no directory is specified, the domains are built under the current directory. The reader is invited to examine the contents of `~/tmp/crewplanning` and `~/tmp/turnandopen`. As it can be seen, both directories consist of a number of testsets that contains pairs of PDDL domain and problem files.

Building planners

However, most people might find it even more useful just to download and compile particular planners. This is done with the module *buildplanner.py*. In our running example, the planner selected is **yahsp2**. Thus, the following command automates all the necessary tasks:

```
 $ ./buildplanner.py --track tempo --subtrack sat
                     --planner yahsp2 yahsp2-mt --directory ~/tmp

[Info: using the svn bookmark specified in the ipc ini configuration file]
[      svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data]


[Info: the following planners have been built]
[      [u'yahsp2', u'yahsp2-mt']]
```

This command explicitly requests downloading all the software of the temporal satisficing planners **yahsp2** and **yahsp2-mt** [6]. Since no svn bookmark has been specified with the directive *--bookmark*, it is automatically retrieved from the INI configuration file `~/.ipc.ini` as reported in the first info message [7]. Finally, the building process of the selected planners finishes and this is reported in the last info message.

[6] Note that because planners and domains are distinguished by the track/subtrack they belong to, planners and domains in differen tracks/subtracks can be named the same.

[7] This actually means that one could download domains from one svn server and planners from a different one. To override the default selection set in the INI configuration file it suffices with specifying a new svn bookmark with the `--bookmark` directive

Upon completion, the reader is invited to check the contents of the directories `~/tmp/yahsp2` and `~/tmp/yahsp2-mt`. There are two files of particular interest `build-yahsp2.log` and `build-yahsp2-mt.log` [8]. These files record the output of all the building process. A quick examination of their contents reveals that this software needs the GNU Multiple Precision library and the flex and bison utilities [9]. To install them:

```
$ sudo apt-get install libgmp3-dev
$ sudo apt-get install bison
$ sudo apt-get install flex
```

Now, before trying to rebuild both planners make sure that the target directories are empty:

```
$ rm -rf ~/tmp/yahsp2*
```

And now issue the same command shown above to build the planners.

Automating the whole process

In general, most users will use this software to automate both tasks: downloading a particular set of domains and build a particular set of planners and to run the planner in all the testsets generated so far. The results shall be stored in a directory which follows the design of a results directory —see *The results directory*. All of these tasks are automated with the script *invokeplanner.py* which uses the previous services. In addition it automates the execution of the planners and stores the results in a particular order as mentioned earlier. Finally, if requested (as in our running example), it sends an automatic e-mail with the log file generated during the whole process.

In case the reader followed the two previous sections, make sure to remove all the directories that contain the planners and domains since `invokeplanner.py` will attempt to create the same directories. If they exist, an error message will be issued and the script resumes execution:

```
$ rm -rf ~/tmp/yahsp2*
$ rm -rf ~/tmp/crewplanning
$ rm -rf ~/tmp/turnandopen
```

Now, the following command starts the whole process:

```
$ ./invokeplanner.py --track tempo --subtrack sat --planner 'yahsp2*'
                     --domain crewplanning turnandopen
                     --directory ~/tmp --logfile tutorial
                     --email carlos.linares@uc3m.es --timeout 10 --memory 2
```

First, note that the script uses much the same directives that are acknowledged by both `buildplanner.py` and `builddomain.py`. In fact they have the same purpose. Other directives which are new in `invokeplanner.py` include:

--logfile This flag requests the script to record the log of all the process in a separate logfile. Since there might be different logfiles that result from different invocations of `invokeplanner.py` the current date and time is appended to the name provided.

--email If an e-mail account was set-up in the INI configuration file, it is allowed now to instruct `invokeplanner.py` to send a copy of the logfile generated to a number of recipients.

--timeout, --memory set the computational resources available. Time is measured in seconds and memory in Gigabytes. If any of these limits is exceeded,

[8] Other names can be used with the option `--logfile`

[9] Besides, the building process complies about a number of **autotools**. But these are not required (as informed by the warning message issued) unless the `Makefile.am` is likely to change and that is not the case here.

the script automatically kills the planner and all of its children, if any.

Upon completion, the script leaves in the directory `~/tmp/results` the results of all the experiments. For a thorough discussion of the purpose and contents of each file generated see *invokeplanner.py*. All the other directories can be safely removed if desired.

Finally, an automatic message gets to the inbox of the specified recipients. Among other important messages, a summary of the results is provided at the bottom. More importantly, the following table reports the number of problems solved so far:

```
 * Number of solved instances:
+-----------+-------------+-------------+-------+
| *         | crewplanning | turnandopen | total |
+-----------+-------------+-------------+-------+
| yahsp2    |      20      |      19     |   39  |
| yahsp2-mt |      20      |      19     |   39  |
| total     |      40      |      38     |       |
+-----------+-------------+-------------+-------+
```

As it can be seen, the planner reports that all planning tasks but one have been solved. If a logfile was requested when invoking `invokeplanner.py`, it is attached to the automated e-mail as well. For a discussion of the information contained in the log file see *The results directory*.

Validating the results

Since all planners are expected to generate plans in a format recognizable by VAL it makes sense to check the validity of all plans generated so far. Since version 1.1 of the software of the Seventh International Planning Competition the version used is VAL-4.2.09 and it can be downloaded from http://www.plg.inf.uc3m.es/ipc2011-deterministic/FrontPage/Software

Before moving on, download this software and install it in your computer as follows:

```
$ tar zxvf VAL-4.2.09.tar.gz
  ...
$ cd VAL-4.2.09
$ make
```

make sure to add the path to the resulting executable `validate` in your `.cshrc` or `.bashrc` configuration files so that they can be executed from any location.

Now, from the directory where the module `IPCData` is located use the script `validate.py` (see *validate.py*) as follows:

```
$ ./validate.py --directory ~/tmp/results
  ...

  +---------------+----------+------------------+--------------------+
  | # directories | # solved | # solution files | # successful plans |
  +---------------+----------+------------------+--------------------+
  |       80      |    40    |       309        |         60         |
  +---------------+----------+------------------+--------------------+
```

This script automatically traverses the results directory and applies the VAL tool to all planning tasks. It then leaves a validation log file in each directory visited. As a result it prints out the summary shown above:

directories informs about the number of directories visited. Since there were two planners running twenty different cases from two different domains, there were 80 directories in total.

solved is the number of plans that have been properly validated. From here it resuls that half of them where either not generated or invalid.

solution files is the total number of plan solution files generated. There were in total up to 309 `plan.soln` files generated —for more information on these files see *invokeplanner.py*. Clearly there were a number of planning tasks that got many solutions.

successful plans only 60 out of the 309 plan solution files generated so far were valid. All the rest being invalid.

7.1.3 Reporting

Once the results have been generated, it is feasible now to inspect them. This is accomplished with the package `IPCReport.py` —see *IPCReport*. It serves not only to inspect the values of particular variables, but also to rank the performance of all the planners at any time within the current limit and also to study how the score of each planner evolves over time. All of these steps are examined in detail in the following subsections.

In the next sections it is assumed that the results directory generated in the previuos section has been moved to another computer with all the necessary facilities (mainly LibreOffice or OpenOffice and a full installation of the LaTeX packages) or that they are available in the same computer where all the previous experimentation was conducted.

Inspecting variables

The module `report.py` (see *report.py*) provides a clean interface to access the values of a large number of variables. For a complete list of all the variables acknowledge by `report.py` see *Reporting variables* or use the command-line argument *--variables*. In the following, a number of examples of its usage are shown.

It would be natural to start wondering what plan solution files were valid and which were not in the previous example. The following command returns a summary of the total number of problems, the total number of problems that are claimed to have been solved by each planner and the total number of problems that have been properly validated:

```
$ ./report.py --directory ~/tmp/results/tempo-sat --variable numprobs numsolved oknumsolved

  ...

name: report
+----------+-----------+-------------+
| numprobs | numsolved | oknumsolved |
+----------+-----------+-------------+
|    80    |     78    |      40     |
+----------+-----------+-------------+
legend:
  numprobs: total number of problems [elaborated data]
  numsolved: number of solved problems (independently of the solution files generated)
             [elaborated data]
  oknumsolved: number of *successfully* solved problems (independently of the solution
               files generated) [elaborated data]

created by IPCrun 1.2 (Revision: 295), Fri Oct  7 15:54:05 2011
```

This information, obtained in a different way, is consistent with the information reported by the logfile sent by `invokeplanner.py` and the summary provided by the `validate.py` module. Note that the directory provided this time is `~/tmp/results/tempo-sat`. When using the reporting tools it is mandatory to provide the root of the track/subtrack and not the root of the results tree.

To get a detailed picture of what really goes on in each particular case:

```
$ ./report.py --directory ~/tmp/results/tempo-sat --variable numsols oknumsols

  ...

name: report
+-----------+--------------+---------+---------+-----------+
|  planner  |    domain    | problem | numsols | oknumsols |
+-----------+--------------+---------+---------+-----------+
|   yahsp2  | crewplanning |   000   |    1    |     1     |
|   yahsp2  | crewplanning |   001   |    1    |     1     |
|   yahsp2  | crewplanning |   002   |    1    |     1     |
|   yahsp2  | crewplanning |   003   |    2    |     2     |
|   yahsp2  | crewplanning |   004   |    2    |     2     |
|   yahsp2  | crewplanning |   005   |    1    |     1     |
|   yahsp2  | crewplanning |   006   |    1    |     1     |
|   yahsp2  | crewplanning |   007   |    1    |     1     |
|   yahsp2  | crewplanning |   008   |    1    |     1     |
|   yahsp2  | crewplanning |   009   |    1    |     1     |
|   yahsp2  | crewplanning |   010   |    3    |     3     |
|   yahsp2  | crewplanning |   011   |    3    |     3     |
|   yahsp2  | crewplanning |   012   |    1    |     1     |
|   yahsp2  | crewplanning |   013   |    1    |     1     |
|   yahsp2  | crewplanning |   014   |    1    |     1     |
|   yahsp2  | crewplanning |   015   |    1    |     1     |
|   yahsp2  | crewplanning |   016   |    1    |     1     |
|   yahsp2  | crewplanning |   017   |    1    |     1     |
|   yahsp2  | crewplanning |   018   |    1    |     1     |
|   yahsp2  | crewplanning |   019   |    3    |     3     |
|   yahsp2  | turnandopen  |   000   |    6    |     0     |
|   yahsp2  | turnandopen  |   001   |    2    |     0     |
|   yahsp2  | turnandopen  |   002   |    6    |     0     |
|   yahsp2  | turnandopen  |   003   |    3    |     0     |
|   yahsp2  | turnandopen  |   004   |    7    |     0     |
|   yahsp2  | turnandopen  |   005   |    11   |     0     |
|   yahsp2  | turnandopen  |   006   |    5    |     0     |
|   yahsp2  | turnandopen  |   007   |    9    |     0     |
|   yahsp2  | turnandopen  |   008   |    8    |     0     |
|   yahsp2  | turnandopen  |   009   |    4    |     0     |
|   yahsp2  | turnandopen  |   010   |    3    |     0     |
|   yahsp2  | turnandopen  |   011   |    0    |     0     |
|   yahsp2  | turnandopen  |   012   |    7    |     0     |
|   yahsp2  | turnandopen  |   013   |    5    |     0     |
|   yahsp2  | turnandopen  |   014   |    10   |     0     |
|   yahsp2  | turnandopen  |   015   |    2    |     0     |
|   yahsp2  | turnandopen  |   016   |    11   |     0     |
|   yahsp2  | turnandopen  |   017   |    5    |     0     |
|   yahsp2  | turnandopen  |   018   |    7    |     0     |
|   yahsp2  | turnandopen  |   019   |    5    |     0     |
| yahsp2-mt | crewplanning |   000   |    1    |     1     |
| yahsp2-mt | crewplanning |   001   |    1    |     1     |
| yahsp2-mt | crewplanning |   002   |    1    |     1     |
| yahsp2-mt | crewplanning |   003   |    4    |     4     |
| yahsp2-mt | crewplanning |   004   |    4    |     4     |
| yahsp2-mt | crewplanning |   005   |    1    |     1     |
| yahsp2-mt | crewplanning |   006   |    1    |     1     |
| yahsp2-mt | crewplanning |   007   |    1    |     1     |
| yahsp2-mt | crewplanning |   008   |    1    |     1     |
| yahsp2-mt | crewplanning |   009   |    1    |     1     |
```

```
| yahsp2-mt | crewplanning |   010   |    3    |     3     |
| yahsp2-mt | crewplanning |   011   |    3    |     3     |
| yahsp2-mt | crewplanning |   012   |    1    |     1     |
| yahsp2-mt | crewplanning |   013   |    1    |     1     |
| yahsp2-mt | crewplanning |   014   |    1    |     1     |
| yahsp2-mt | crewplanning |   015   |    1    |     1     |
| yahsp2-mt | crewplanning |   016   |    1    |     1     |
| yahsp2-mt | crewplanning |   017   |    1    |     1     |
| yahsp2-mt | crewplanning |   018   |    1    |     1     |
| yahsp2-mt | crewplanning |   019   |    3    |     3     |
| yahsp2-mt | turnandopen  |   000   |    23   |     0     |
| yahsp2-mt | turnandopen  |   001   |    3    |     0     |
| yahsp2-mt | turnandopen  |   002   |    11   |     0     |
| yahsp2-mt | turnandopen  |   003   |    3    |     0     |
| yahsp2-mt | turnandopen  |   004   |    5    |     0     |
| yahsp2-mt | turnandopen  |   005   |    7    |     0     |
| yahsp2-mt | turnandopen  |   006   |    3    |     0     |
| yahsp2-mt | turnandopen  |   007   |    3    |     0     |
| yahsp2-mt | turnandopen  |   008   |    1    |     0     |
| yahsp2-mt | turnandopen  |   009   |    1    |     0     |
| yahsp2-mt | turnandopen  |   010   |    1    |     0     |
| yahsp2-mt | turnandopen  |   011   |    0    |     0     |
| yahsp2-mt | turnandopen  |   012   |    11   |     0     |
| yahsp2-mt | turnandopen  |   013   |    8    |     0     |
| yahsp2-mt | turnandopen  |   014   |    12   |     0     |
| yahsp2-mt | turnandopen  |   015   |    12   |     0     |
| yahsp2-mt | turnandopen  |   016   |    6    |     0     |
| yahsp2-mt | turnandopen  |   017   |    9    |     0     |
| yahsp2-mt | turnandopen  |   018   |    11   |     0     |
| yahsp2-mt | turnandopen  |   019   |    3    |     0     |
+-----------+--------------+---------+---------+-----------+
legend:
  planner [key]
  domain [key]
  problem [key]
  numsols: total number of solution files generated [raw data]
  oknumsols: total number of *successful* solution files generated [raw data]

created by IPCrun 1.2 (Revision: 295), Fri Oct  7 15:58:09 2011
```

This command reports the number of plan solution files generated and how many of them were valid. As specified in the logfile sent by `invokeplanner.py` both planners failed to solve a particular instance of the domain turnandopen. This instance is *011*. Besides, though both planners generated a number of plan solution files in the turnandopen domain, none is valid.

Whether both planners failed on time or memory or due to any other reason can be queried also:

```
$ ./report.py --directory ~/tmp/results/tempo-sat
              --variable numfails timefails memfails unexfails

  ...

name: report
+----------+----------------+----------+-----------+
| numfails |   timefails    | memfails | unexfails |
+----------+----------------+----------+-----------+
|    2     | ['011', '011'] |    []    |     []    |
+----------+----------------+----------+-----------+
```

```
legend:
  numfails: total number of fails [elaborated data]
  timefails: problem ids where the planner failed on time [elaborated data]
  memfails: problem ids where the planner failed on memory [elaborated data]
  unexfails: problem ids where the planner unexpectedly failed [elaborated data]

created by IPCrun 1.2 (Revision: 295), Fri Oct  7 16:03:57 2011
```

As it can be seen, both planners could not solve problem *011* in the given time limit so that they were killed after the time bound (10 seconds) was reached.

It is also feasible to examine the performance of the planner from a different point of view. For example, to examine the memory profile of the planner when solving a particular problem. Take problem *011* of the crewplanning domain and examine the memory usage of **yahsp2**:

```
$ ./report.py --directory ~/tmp/results/tempo-sat
              --variable timelabels memlabels --unroll
              --planner 'yahsp2$' --domain crewplanning --problem 011

  ...

name: report
+-----------+--------------+---------+------------+-----------+
|  planner  |    domain    | problem | timelabels | memlabels |
+-----------+--------------+---------+------------+-----------+
|   yahsp2  | crewplanning |   011   |    4.84    |   22.72   |
|   yahsp2  | crewplanning |   011   |    9.72    |   34.97   |
+-----------+--------------+---------+------------+-----------+
legend:
  planner [key]
  domain [key]
  problem [key]
  timelabels: time ticks used for sampling memory consumption (in seconds) [raw data]
  memlabels: memory consumption at a particular time tick (in MB) [raw data]

created by IPCrun 1.2 (Revision: 295), Fri Oct  7 16:12:18 2011
```

Since the *--planner* directive accepts arbitrary regular expressions, it is necessary to add `$` at the end to avoid the matcher to accept the string `yahsp2-mt`. The same report can be generated in GNU Octave format and the output redirected to a file:

```
$ ./report.py --directory ~/tmp/results/tempo-sat --variable timelabels memlabels
              --unroll --planner 'yahsp2$' --domain crewplanning --problem 011
              --style octave --quiet > output.m
```

The directive *--quiet* has been added to avoid having the preamble in the resulting file. This way, `output.m` can be easily ingested by either GNU Octave or gnuplot. The resulting file is shown below:

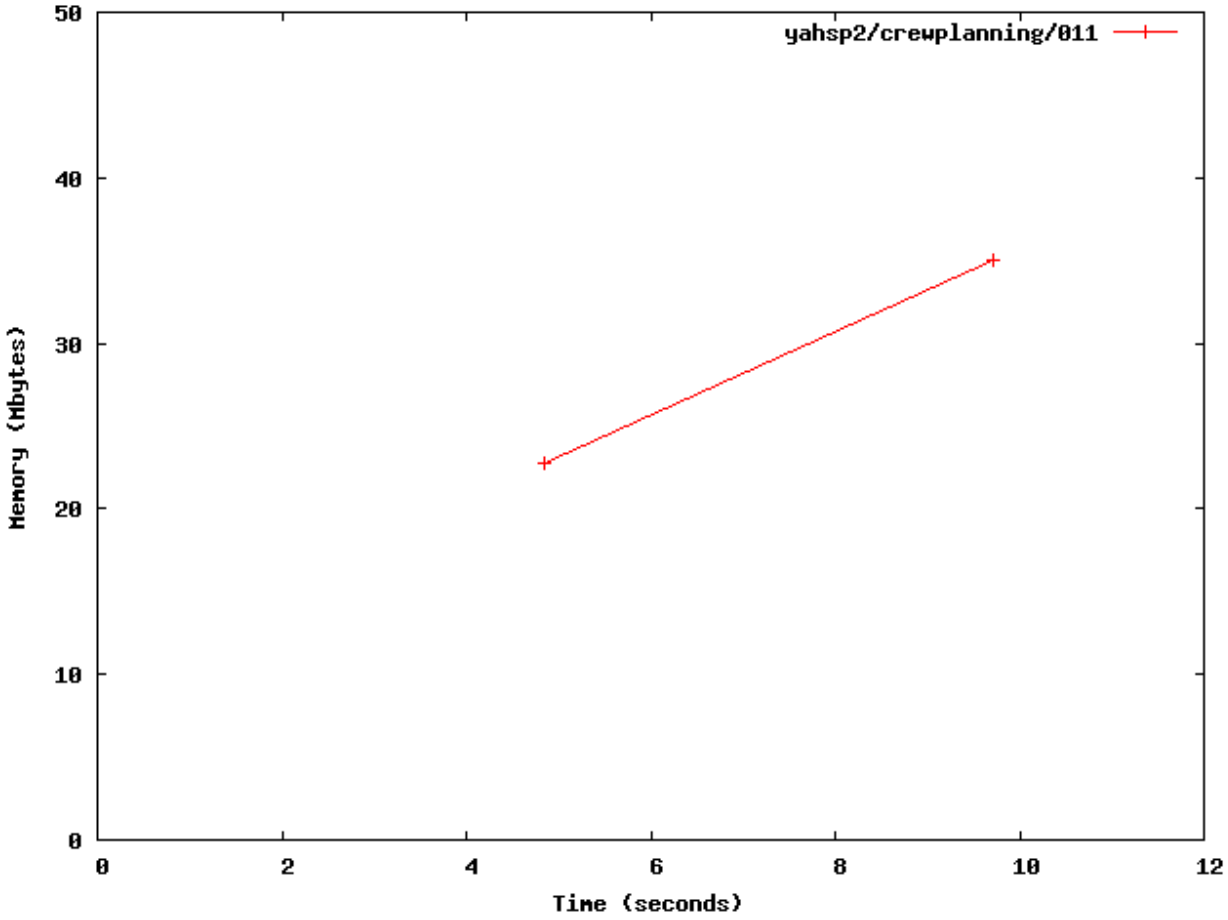
```
# created by IPCrun 1.2 (Revision: 295), Fri Oct  7 16:17:50 2011
# name: report
# type: matrix
# rows: 2
# columns: 5
yahsp2 crewplanning 011 4.84 22.72
yahsp2 crewplanning 011 9.72 34.97
# legend:
#   planner [key]
#   domain [key]
#   problem [key]
```

```
#   timelabels: time ticks used for sampling memory consumption (in seconds) [raw data]
#   memlabels: memory consumption at a particular time tick (in MB) [raw data]
```

Now, this file can be used in a session with gnuplot as follows:

```
gnuplot> set xlabel "Time (seconds)"
gnuplot> set ylabel "Memory (Mbytes)"
gnuplot> set terminal png
gnuplot> set output "memprofile.png"
gnuplot> plot [0:12] [0:50] "output.m" using 4:5 with linesp title "yahsp2/crewplanning/011"
```

The resulting image is shown below:



As another example, let us examine now how the quality of plans improved with new ones and how it compares to the number of actions in each plan. The following command requests `report.py` to show the quality and length of validated plans as reported by VAL for the planner **yahsp2-mt** in the planning task *003* of the domain crewplanning:

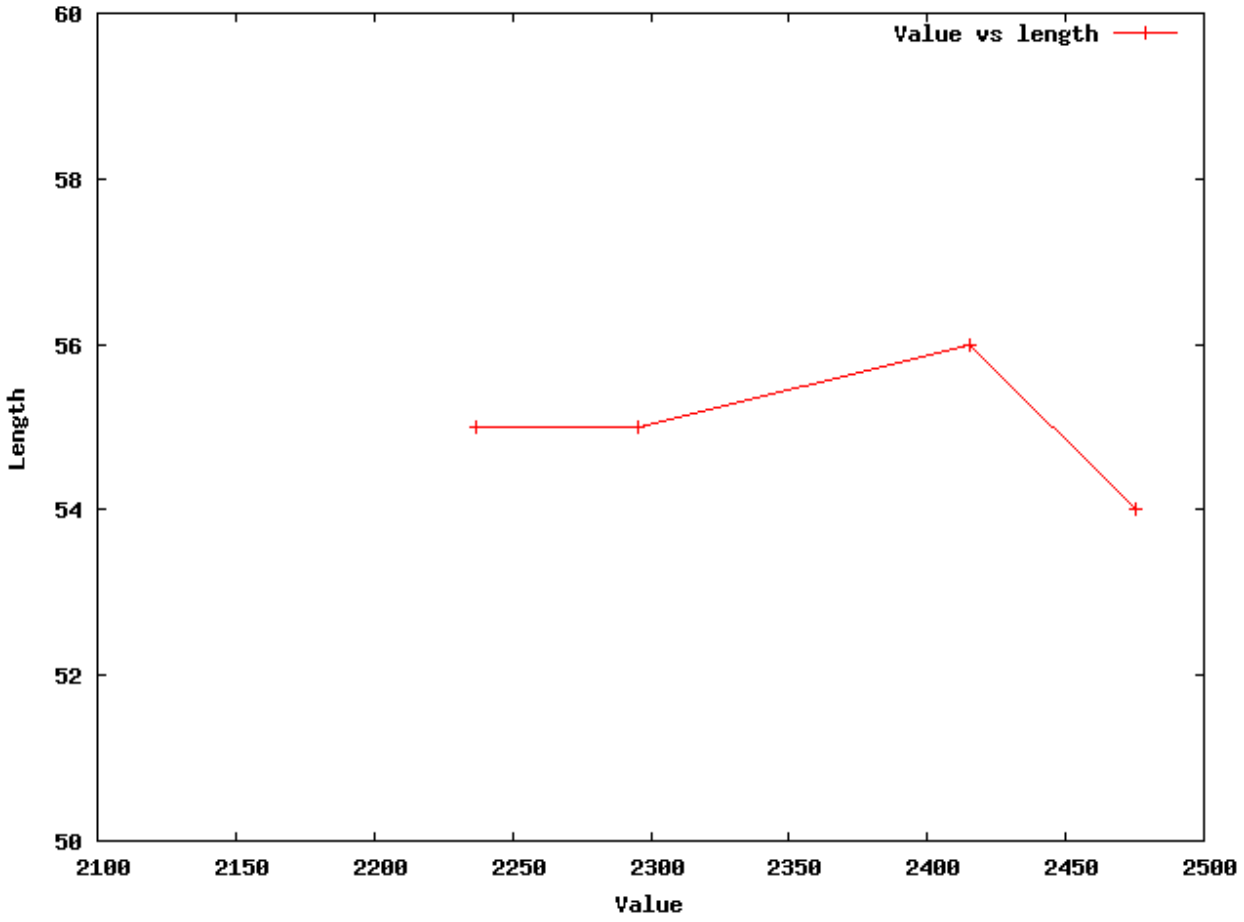
```
$ ./report.py --directory ~/tmp/results/tempo-sat/ --variable values lengths
              --planner yahsp2-mt --domain crewplanning --problem 003 --unroll
              --style octave --quiet > output.m
```

Now, the resulting file can be ingested in a gnuplot session as follows:

```
gnuplot> set xlabel "Value"
gnuplot> set ylabel "Length"
gnuplot> set terminal png
```

```
gnuplot> set output "length.png"
gnuplot> plot [2100:2500] [50:60] "output.m" using 4:5 with linesp title "Value vs length"
```

The resulting image is shown below:

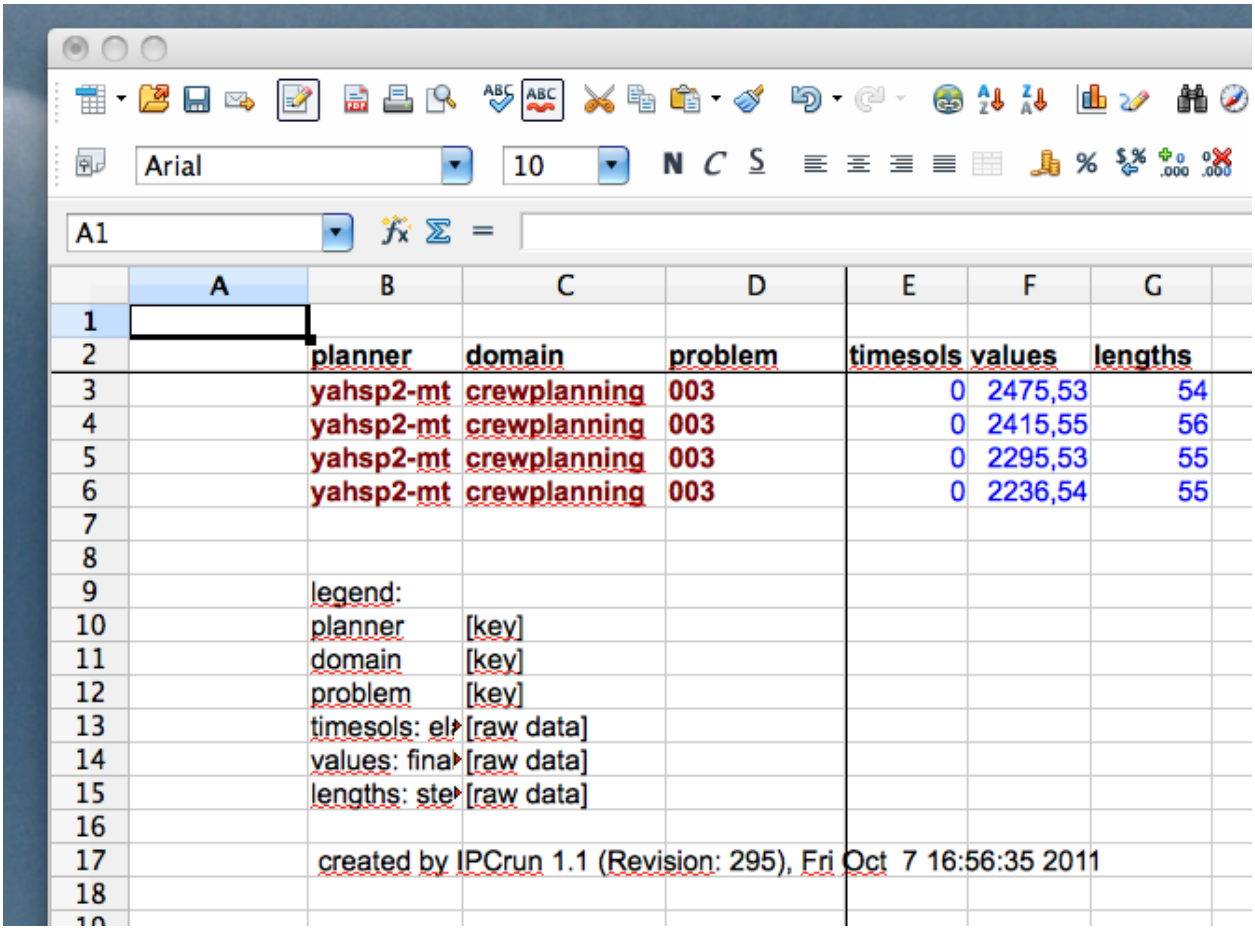


Since Gnuplot plots the points in ascending order, these values have to be read in reversed order, the first solution being the rightmost point and the last one being the leftmost one. If the same information would be needed but with the exact timings when each solution was generated, the following command makes the service:

```
$ ./report.py --directory tempo-sat/ --variable timesols values lengths
              --planner yahsp2-mt --domain crewplanning --problem 003
              --unroll --style excel
```

This time, the report is generated in an Excel wordsheet named `report.xls`. A screenshot (generated with OpenOffice [10]) is shown below:

[10] Nowadays, LibreOffice is a nice alternative to OpenOffice.



	A	B	C	D	E	F	G
1							
2		planner	domain	problem	timesols	values	lengths
3		yahsp2-mt	crewplanning	003	0	2475,53	54
4		yahsp2-mt	crewplanning	003	0	2415,55	56
5		yahsp2-mt	crewplanning	003	0	2295,53	55
6		yahsp2-mt	crewplanning	003	0	2236,54	55
7							
8							
9		legend:					
10		planner	[key]				
11		domain	[key]				
12		problem	[key]				
13		timesols: el	[raw data]				
14		values: fina	[raw data]				
15		lengths: ste	[raw data]				
16							
17		created by IPCrun 1.1 (Revision: 295), Fri Oct 7 16:56:35 2011					
18							

Ranking planners

It is usually a good idea, not only for the International Planning Competition, but also at the lab when conducting experiments with a number of planners to rank them according to their performance with regard to a particular set of planning tasks. The modules `score.py` (see *score.py*) and `tscore.py` (see *tscore.py*) are provided to get both an overall view and also a detailed picture of the performance of each planner in the results tree under consideration.

We start by showing a nice feature of the reporting tools. In all cases discussed in the previous subsection, the `report.py` module examined a given directory. It forced it to traverse the whole tree structure, visiting the directories of all the planning tasks to extract some information and then to back it up all the tree upwards. Because this tree contained only a few planners and domains, they could be processed very fast. However, if the tree gets large, this can result in an unecessary waste of time. To avoid it, summaries or snapshots can be generated instead —see *Snapshots*. These are just binary files that contain all the necessary information. Snapshots can be only generated by the module `report.py`. It suffices with requesting a report while specifing a snapshot filename as follows:

```
$ ./report.py --directory tempo-sat/ --variable numprobs
              --summarize tutorial.snapshot
```

Although only one variable has been specified, the resulting snapshot contains the values of all the acknowledged variables —see *Reporting variables*. Now, this snapshot can be used either by any of the modules in the package `IPCReport`. In this subsection, its usage is shown when ranking planners.

To rank planners using the metric used at the Seventh International Planning Competition with the snapshot previously generated [11]:

[11] These tables were inspired by a very similar ones that were automatically generated in the Sixth International Planning Competition.

```
$ ./score.py --summary tutorial.snapshot

  ...

tempo-sat: crewplanning (Fri Oct  7 17:26:46 2011)
+-------+--------+-----------+---------+
|  no.  | yahsp2 | yahsp2-mt |   best  |
+-------+--------+-----------+---------+
|  000  |  1.00  |    1.00   | 2880.19 |
|  001  |  1.00  |    1.00   | 2880.20 |
|  002  |  1.00  |    1.00   | 2880.25 |
|  003  |  0.93  |    1.00   | 2236.54 |
|  004  |  1.00  |    1.00   | 1755.34 |
|  005  |  1.00  |    1.00   | 4320.39 |
|  006  |  1.00  |    1.00   | 4320.38 |
|  007  |  1.00  |    1.00   | 4320.48 |
|  008  |  1.00  |    1.00   | 3450.64 |
|  009  |  1.00  |    1.00   | 4230.97 |
|  010  |  1.00  |    1.00   | 5087.05 |
|  011  |  1.00  |    1.00   | 5087.16 |
|  012  |  1.00  |    1.00   | 5926.16 |
|  013  |  1.00  |    1.00   | 6706.43 |
|  014  |  1.00  |    1.00   | 7126.77 |
|  015  |  1.00  |    1.00   | 7486.85 |
|  016  |  1.00  |    1.00   | 3810.70 |
|  017  |  1.00  |    1.00   | 3990.76 |
|  018  |  1.00  |    1.00   | 4771.09 |
|  019  |  1.00  |    1.00   | 4337.25 |
| total | 19.92  |   20.00   |         |
+-------+--------+-----------+---------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.1 (Revision: 295), Fri Oct  7 17:26:46 2011


tempo-sat: turnandopen (Fri Oct  7 17:26:46 2011)
+-------+--------+-----------+------+
|  no.  | yahsp2 | yahsp2-mt | best |
+-------+--------+-----------+------+
|  000  |   X    |     X     | ---  |
|  001  |   X    |     X     | ---  |
|  002  |   X    |     X     | ---  |
|  003  |   X    |     X     | ---  |
|  004  |   X    |     X     | ---  |
|  005  |   X    |     X     | ---  |
|  006  |   X    |     X     | ---  |
|  007  |   X    |     X     | ---  |
|  008  |   X    |     X     | ---  |
|  009  |   X    |     X     | ---  |
|  010  |   X    |     X     | ---  |
|  011  |  ---   |    ---    | ---  |
|  012  |   X    |     X     | ---  |
|  013  |   X    |     X     | ---  |
|  014  |   X    |     X     | ---  |
|  015  |   X    |     X     | ---  |
|  016  |   X    |     X     | ---  |
```

```
|  017  |   X    |     X     | ---  |
|  018  |   X    |     X     | ---  |
|  019  |   X    |     X     | ---  |
| total |  0.00  |    0.00   |      |
+-------+--------+-----------+------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.1 (Revision: 295), Fri Oct  7 17:26:46 2011


tempo-sat: ranking (Fri Oct  7 17:26:46 2011)
+-----------+--------------+-------------+-------+
|  planner  | crewplanning | turnandopen | total |
+-----------+--------------+-------------+-------+
| yahsp2-mt |    20.00     |     0.00    | 20.00 |
|   yahsp2  |    19.92     |     0.00    | 19.92 |
|   total   |    39.92     |     0.00    |       |
+-----------+--------------+-------------+-------+

 ---: unsolved
  X : invalid

 created by IPCrun 1.1 (Revision: 295), Fri Oct  7 17:26:46 2011
```

As it can be seen, **yahsp2-mt** is slight better (according to the quality metric defined in the Seventh International Planning Competition) and should be preferred for those planning tasks that are similar to those used in this tutorial.

The same information can be generated in a LaTeX file that can be easily processed to get a pdf:

```
$ ./score.py --summary tutorial.snapshot --style latex
```

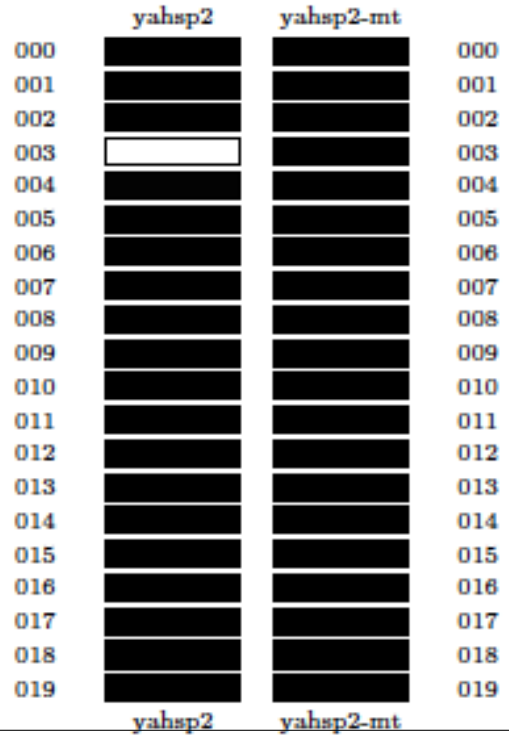
The resulting file is stored in a file named `matrix.tex`. This file cannot be processed with **pdflatex** since it uses the ps-tricks package. Instead, the `IPCReport` provides a `makefile` that can be used to generate the pdf:

```
$ make matrix.pdf
```

The resulting pdf file contains as many pages as domains have been found in the snapshot plus an additional one with an overall ranking table that summarizes all results. The following pictures show the three pages that are generated by the previous command:

tempo-sat: crewplanning (Fri Oct 7 17:33:39 2011)

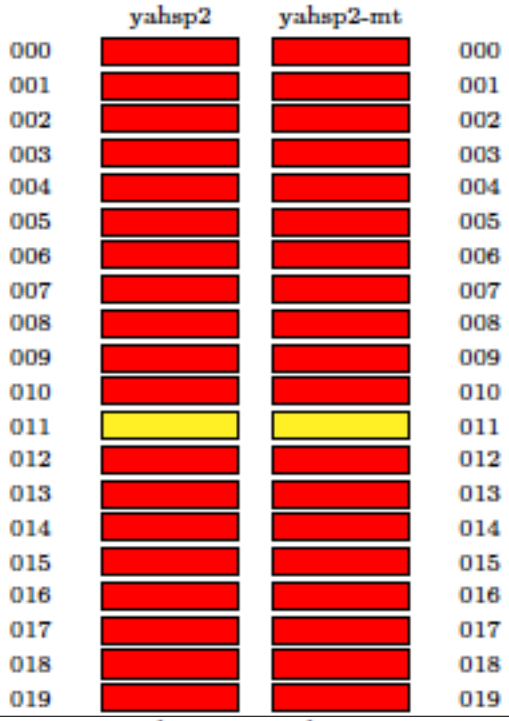
no.	yahsp2	yahsp2-mt	best
000	1.00	1.00	2880.19
001	1.00	1.00	2880.20
002	1.00	1.00	2880.25
003	0.93	1.00	2236.54
004	1.00	1.00	1755.34
005	1.00	1.00	4320.39
006	1.00	1.00	4320.38
007	1.00	1.00	4320.48
008	1.00	1.00	3450.64
009	1.00	1.00	4230.97
010	1.00	1.00	5087.05
011	1.00	1.00	5087.16
012	1.00	1.00	5926.16
013	1.00	1.00	6706.43
014	1.00	1.00	7126.77
015	1.00	1.00	7486.85
016	1.00	1.00	3810.70
017	1.00	1.00	3990.76
018	1.00	1.00	4771.09
019	1.00	1.00	4337.25
total	19.92	20.00	



The first page shows the results in the crewplanning domain. The color codes used in the lower half represent the same information shown in the upper half but are easier to understand. The darker a box the better the performance; likewise, the lighter the box the worse. Here, better and worse are referred to the relative performance of each planner with respect to the others.

tempo-sat: turnandopen (Fri Oct 7 17:33:39 2011)

no.	yahsp2	yahsp2-mt	best
000	✘	✘	∅
001	✘	✘	∅
002	✘	✘	∅
003	✘	✘	∅
004	✘	✘	∅
005	✘	✘	∅
006	✘	✘	∅
007	✘	✘	∅
008	✘	✘	∅
009	✘	✘	∅
010	✘	✘	∅
011	∅	∅	∅
012	✘	✘	∅
013	✘	✘	∅
014	✘	✘	∅
015	✘	✘	∅
016	✘	✘	∅
017	✘	✘	∅
018	✘	✘	∅
019	✘	✘	∅
total	0.00	0.00	

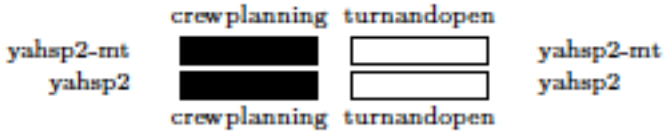


The second page shows the results in the turnandopen domain. In this case, the color codes are only either red or yellow. A red entry means that at least one plan was generated but that none was found valid. A yellow box means that no solution was generated for that particular problem.

In both cases, rows and columns are sorted lexicographically.

tempo-sat: ranking (Fri Oct 7 17:33:39 2011)

planner	crewplanning	turnandopen	total
yahsp2-mt	20.00	0.00	**20.00**
yahsp2	19.92	0.00	**19.92**
total	**39.92**	**0.00**	



The third page provides an overall view of the performance of each planner in every domain. In this case, planners and domains are ranked in descending order of their total score. The color codes shown at the bottom follow the same notation discussed above.

Finally, a demo of the `tscore.py` is skipped since a full example is shown in *tscore.py*.

7.2 Second practical case

The second example shows how to set-up the environment for running experiments with the software developed at the Seventh International Planning Competition. This example has been explicitly created to encourage users to use this

software for evaluating their own planners. The exercise consists of just creating a private svn server that will host just one planner and five problems of only one domain.

7.2.1 Creating the SVN repository

Before starting with this part of the tutorial, move to an empty ancilliary directory and make sure to follow all the steps depicted in *Setting up* but *Configuring the IPC Software* —if the external software described in those sections is already installed in your computer you can safely skip those steps. Next, create a new directory as follows:

```
$ cd ~/tmp
$ mkdir second-case
$ cd second-case
```

In this case two svn servers will be used at the same time: on one hand, the scripts will be retrieved from the official svn server of the Seventh International Planning Competition; on the other hand, a private svn server will be created to store an experimental planner and one testing domain. Therefore, if you decide to configure the IPC software as explained in *Configuring the IPC Software* make sure to use the following bookmark: `file:///.../tmp/tutorial` where the ellipsis should be replaced by the absolute path to your temporal directory [12] —more on this below.

First things first. This part of the tutorial will use the planner **probe** and the first five problems of the domain openstacks. The planner **probe** is known to have two specific requirements: **makedepend** and SCons so make sure to make them available in your GNU/Linux box:

```
$ sudo apt-get install xutils-dev
$ sudo apt-get install scons
```

Of course, it is expected that the user replaces this planner and domain by those of her choice but they are used here to make sure that they are available for private experimentation. Both are referred to the sequential satisficing (`seq-sat`) track of the Seventh International Planning Competition. Thus, it is mandatory to recreate in the local computer the same structure that shall be followed in the svn repository so that the IPC software can handle it —for more information see *Contents of the data repository*:

```
$ mkdir -p my-ipc/planners/seq-sat
$ mkdir -p my-ipc/domains/seq-sat
$ svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/planners/seq-sat/seq-sat-probe
      my-ipc/planners/seq-sat/seq-sat-probe
$ svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/domains/seq-sat/openstacks
      my-ipc/domains/seq-sat/openstacks
```

Before moving on, keep an eye on a very important issue. To make it feasible to automate both the building process of the planner and the creation of the testsets, the user has to provide to very important files: `plan` and `build` —for more details see *check.py*. The first one acts as a wrapper to command the planner to solve a particular problem in a particular domain. The second one shall fully automate the building process of the software. Make sure to:

- Prefix these scripts with appropriate shebang lines so that they can be executed
- Add all the necessary commands in `plan` to make your planner run in the desired mode
- Avoid your script `build` to use sudo command lines

These files are mandatory. Without them, the whole process will fail. It is the responsiblity of the programmer to provide them as the author of **probe** (Nir Lipovetzky) did here. When running your own example make sure to provide them. It might be a good idea to test these scripts locally before going any further.

Since we only want the first five problems of the domain openstacks, remove all the rest:

[12] In my case this would be `/home/clinares` so that the whole svn bookmark reads as `file:///home/clinares/tmp/tutorial` and it should be pretty similar in all cases.

```
$ cd my-ipc/domains/seq-sat/openstacks
$ rm -rf domain/p06* domain/p07* domain/p08* domain/p09*
         domain/p1* domain/p2*
$ rm -rf problems/p06* problems/p07* problems/p08* problems/p09*
         problems/p1* problems/p2*
```

Finally, since all these files will go into a new repository, make sure to remove all the `.svn` subdirectories as follows:

```
$ cd ../../../
$ find . -name ".svn" -exec rm -rf \{\} \;
```

Now, everything is ready to create a new repository that will hold the selected planner and domain. First, create a svn repository called `tutorial` in your temporal directory as follows:

```
$ cd ~/tmp
$ svnadmin create tutorial
```

If you are not familiar with svn you are very welcome to examine the contents of this repository by browsing its directories. Bear in mind that this svn server is local and therefore will be accessed with the bookmark prefix `file://`. Since the path to it is `.../tmp/tutorial` (where the ellipsis stand for the absolute path to your home directory) a svn bookmark such as `file:///.../tmp/tutorial` results —note the three slashes after `file`: two are required by the svn prefix whereas the third one indicates the beginning of an absolute path. Now, check in the contents of `my-ipc` into the newly created repository [13]:

```
$ svn import second-case/my-ipc file:///home/clinares/tmp/tutorial -m "Initial import"
```

You are very welcome to browse the contents of your repository with **svn list**. For example:

```
$ svn list file:///home/clinares/tmp/tutorial
domains/
planners/
$ svn list file:///home/clinares/tmp/tutorial/domains
seq-sat/
```

You can inspect the contents of any directory in the repository just by adding it to the svn bookmark. Verify that the contents of your svn server are compliant with the arrangement described in *Contents of the data repository*.

Now that the svn repository has been created with the desired contents, you can safely remove your local directory:

```
$ rm -rf ~/tmp/second-case
```

Do not be scared! Its contents will be timely retrieved as shown below.

7.2.2 Testing the new SVN repository

The curious reader might want to try to automatically build the testsets of the domain stored in the new svn server or to automatically build the planner **probe** (while being in the directory where the package `IPCData` was checked-out):

```
$ ./buildplanner.py --track seq --subtrack sat --planner probe --directory ~/tmp
                    --bookmark file:///home/clinares/tmp/tutorial
$ ./builddomain.py --track seq --subtrack sat --domain openstacks --directory ~/tmp
                   --bookmark file:///home/clinares/tmp/tutorial
```

Note that this time a bookmark has been provided. The reason is that this svn server has not been configured as in the first practical case —see *Configuring the IPC Software*.

[13] `/home/clinares` has been added for the sake of clarity but the absolute path to your temporal directory shall replace it

However, it is usually more practical to use the script `invokeplanner.py` instead as follows (if you typed in the previous two commands make sure to remove the directories `~/tmp/probe` and `~/tmp/openstacks` before proceeding):

```
$ ./invokeplanner.py --track seq --subtrack sat --planner probe --domain openstacks
                     --directory ~/tmp --bookmark file:///home/clinares/tmp/tutorial
                     --timeout 30 --memory 2
```

Since this svn server has not been configured, automatic notification by e-mail is not allowed. Instead, the log file will be shown on the standard output as the experiments progress —if the option *--verbose* is given, more information will be printed out. The information shown on the screen ends with a summary report of the number of problems solved and the number of plan solution files generated. In my computer (a rather old one), probe solved three of them generating up to 17 plan solution files.

Before making any queries to the results tree, validate all the results (make sure to have already installed in your system VAL 4.2.09 as discussed in *Validating the results*):

```
$ ./validate.py --directory ~/tmp/results/seq-sat
```

Next, a figure different than those shown in the first practical case is created here. To come out with the typical figure that shows the solving times and their corresponding quality two lines are about to be plot that represent the quality of the first and last solution at the corresponding times they are found. First, to make sure that the first solution is the worst one and, likewise, that the last solution is that the best one, command `IPCReport` to show up the qualities of all solutions found and the time stamps when they were computed [14]:

```
$ mv ../IPCReport
$ ./report.py --directory ~/tmp/results/seq-sat/ --variable timesols values --quiet
 name: report
+-----+---------+------------------------------+-------------------------------------------+
| ... | problem |           timesols           |                  values                   |
+-----+---------+------------------------------+-------------------------------------------+
| ... |   000   |    [6, 7, 10, 15, 22, 30]    |    [15.0, 14.0, 13.0, 12.0, 11.0, 10.0]   |
| ... |   001   |       [19, 21, 23, 25]       |          [24.0, 23.0, 22.0, 21.0]         |
| ... |   002   | [16, 18, 20, 22, 24, 26, 30] | [23.0, 22.0, 21.0, 20.0, 19.0, 18.0, 17.0] |
| ... |   003   |              []              |                    []                     |
| ... |   004   |              []              |                    []                     |
+-----+---------+------------------------------+-------------------------------------------+
 legend:
   planner [key]
   domain [key]
   problem [key]
   timesols: elapsed time when each solution was generated (in seconds) [raw data]
   values: final values returned by VAL, one per each *valid* solution file [raw data]

 created by IPCrun 1.2 (Revision: 295), Mon Oct 10 00:37:25 2011
```

Since this hyphotesis is satisfied (and should be satisfied always unless something goes really wrong!) create two octave files with the following commands:

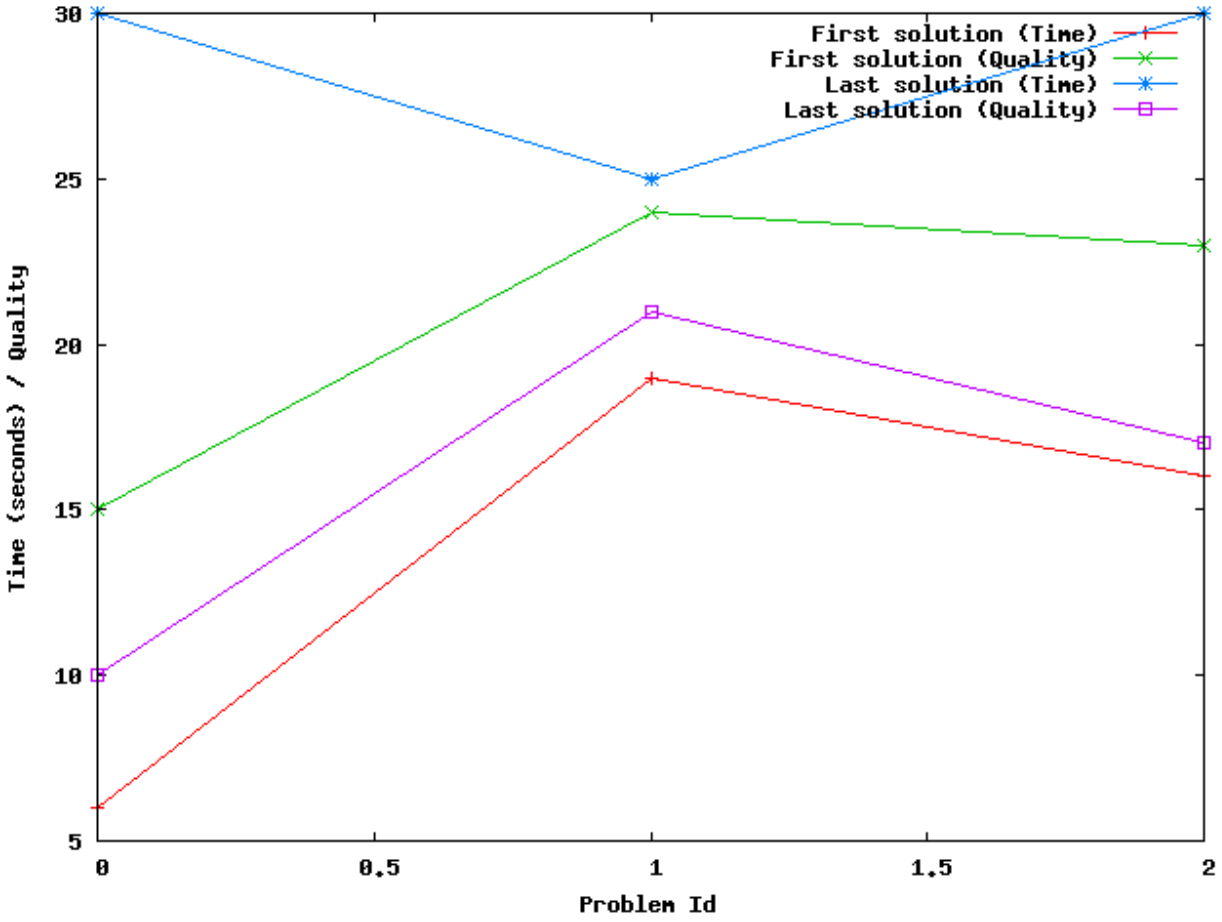
```
$ ./report.py --directory ~/tmp/results/seq-sat/ --variable oktimefirstsol uppervalue --quiet
              --style octave > first.m
$ ./report.py --directory ~/tmp/results/seq-sat/ --variable oktimelastsol lowervalue --quiet
              --style octave > last.m
```

Remove the lines for the last two problems and now start a gnuplot session:

[14] The first two columns have been ommitted to make the whole table fit the width of the LaTeX version of this manual

```
gnuplot> set xlabel "Problem Id"
gnuplot> set ylabel "Time (seconds) / Quality"
gnuplot> set terminal png
gnuplot> set output "timequality.png"
gnuplot> plot "first.m" using 3:4 with linesp title "First solution (Time)",
              "first.m" using 3:5 with linesp title "First solution (Quality)",
              "last.m" using 3:4 with linesp title "Last solution (Time)",
              "last.m" using 3:5 with linesp title "Last solution (Quality)"
```

The resulting image is shown below. It shows simultaneously both the time and quality of all the problems **probe** managed to solve:



7.3 Third practical case

The main goal of this exercise is to show how to perform comparisons among planners. To this end, we will use the results of the previous case to compare its performance with the winner of the Sequential Satisficing (`seq-sat`) track: **lama-2011**. In general, however, the same steps can be used to compare the performance of any planners in any set of domains. Therefore, this procedure can be used to make private experiments —for example, prior to the publication of a paper introducing a new planner.

After the preliminary experiments shown in the previous section, we will extend the experimentation to all problems in the openstacks domain. Therefore, before moving on make sure to follow the same procedure depicted before but without deleting any planning task. Instead, preserve all of them.

7.3.1 Getting the results tree directories

As it turned out in the previous case, we will be simultaneously using two different svn servers: one with the results of a private experimentation (that is exemplified in this case by experiments with **probe**) and another one with the results of an official entrant to the Seventh International Planning Competition.

As soon as a private server has been set-up (with the bookmark `file://`) named tutorial which contains only **probe** and all the planning tasks in the domain openstacks, obtain the new results tree structure as follows by running the module `invokeplanner.py`:

```
$ ./invokeplanner.py --track seq --subtrack sat --planner probe --domain openstacks
                     --directory ~/tmp --bookmark file:///home/clinares/tmp/tutorial
                     --timeout 30 --memory 2
```

Do not forget to validate these results using the module `validate.py` from the package `IPCData`:

```
$ ./validate.py --directory ~/tmp/results/seq-sat/probe/

 Revision: 296
 Date: 2011-10-06 01:16:57 +0200 (Thu, 06 Oct 2011)

 ./validate.py 1.1

 ------------------------------------------------------------------------------
 * directory : /Users/clinares/tmp/third-sat/probe
 ------------------------------------------------------------------------------

+---------------+----------+------------------+--------------------+
| # directories | # solved | # solution files | # successful plans |
+---------------+----------+------------------+--------------------+
|       20      |    4     |        23        |         23         |
+---------------+----------+------------------+--------------------+
```

As it can be seen this planner did not generate any invalid solution.

Now, retrieve from the official svn server of the Seventh International Planning Competition the results of **lama-2011** in the same domain and store them in a separate directory of your temporal folder:

```
$ cd ~/tmp
$ mkdir -p ipc-sat/lama-2011
$ cd ipc-sat/lama-2011
$ svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/results/raw/seq-sat/lama-2011/openstacks
```

Two important remarks follow: first, when retrieving the results of **lama-2011** in the openstacks domain they are stored in a directory which has to adhere to the conventions described in *The results directory* —here *ipc* and *sat* are the names of artificial tracks and subtracks; second, the results checked out from the official svn server are in raw format [15] so that they have to be validated before proceeding. Invoke the validation script from the module IPCData as follows:

```
$ ./validate.py --directory ~/tmp/ipc-sat

Revision: 296 Date: 2011-10-05 16:16:57 -0700 (Wed, 05 Oct 2011)

./validate.py 1.1

------------------------------------------------------------------------------
* directory : /home/clinares/tmp/ipc-sat
```

[15] As a matter of fact, it is feasible instead to check out the validated contents from the directory `val` instead of `raw`. However, it is highly recommended to download always the raw results and to validate them locally.

```
------------------------------------------------------------------------------

+---------------+----------+------------------+--------------------+
| # directories | # solved | # solution files | # successful plans |
+---------------+----------+------------------+--------------------+
|       20      |    20    |       187        |        187         |
+---------------+----------+------------------+--------------------+
```

7.3.2 Combining the results tree directories

Now, since the reporting scripts work from a unique results tree directory it is mandatory to merge both results tree directories into a separate one. In this example, it will be called `third-sat`.

First, start by populating `third-sat` with the contents of the results tree directory of **lama-2011**. To this end, move to the package `IPCTools` and execute the following command:

```
 $ ./copy.py --source ~/tmp/ipc-sat/lama-2011 --destination ~/tmp/third-sat

Revision: 296
Date: 2011-10-06 01:16:57 +0200 (Thu, 06 Oct 2011)

./copy.py 1.1

------------------------------------------------------------------------------
* source      : ['/Users/clinares/tmp/ipc-sat/lama-2011']
* destination : /Users/clinares/tmp/third-sat
------------------------------------------------------------------------------
```

The user is invited to inspect the contents of a particular problem in this artificial results tree directory:

```
$ cd ~/tmp/third-sat/lama-2011/openstacks/013
$ ls -lAF
total 1328
drwxr-xr-x  7 clinares  staff     238 Oct 11 16:09 .svn/
-rw-r--r--  1 clinares  staff    6826 Oct 11 16:09 _third-sat.lama-2011-openstacks.013-cpu
-rw-r--r--  1 clinares  staff   51202 Oct 11 16:09 _third-sat.lama-2011-openstacks.013-log
-rw-r--r--  1 clinares  staff    1266 Oct 11 16:09 _third-sat.lama-2011-openstacks.013-mem
-rw-r--r--  1 clinares  staff     873 Oct 11 16:12 _third-sat.lama-2011-openstacks.013-val
-rw-r--r--  1 clinares  staff     235 Oct 11 16:09 _third-sat.lama-2011-openstacks.013-ver
-rw-r--r--  1 clinares  staff   65513 Oct 11 16:09 build-lama-2011.log
-rw-r--r--  1 clinares  staff   94228 Oct 11 16:09 domain.pddl
-rw-r--r--  1 clinares  staff   14384 Oct 11 16:09 plan.soln.1
-rw-r--r--  1 clinares  staff   14357 Oct 11 16:09 plan.soln.2
-rw-r--r--  1 clinares  staff   14168 Oct 11 16:09 plan.soln.3
-rw-r--r--  1 clinares  staff   14141 Oct 11 16:09 plan.soln.4
-rw-r--r--  1 clinares  staff   14114 Oct 11 16:09 plan.soln.5
-rw-r--r--  1 clinares  staff       0 Oct 11 16:09 planner.err
-rw-r--r--  1 clinares  staff  319586 Oct 11 16:09 planner.log
-rw-r--r--  1 clinares  staff   36603 Oct 11 16:09 problem.pddl
```

As it can be seen, the files have been conveniently renamed to make them look as if they were originally created under a track/subtrack named `third-sat`.

Next, extend the contents of this results directory by copying there the results obtained with the planner **probe**:

```
 $ ./copy.py --source ~/tmp/results/seq-sat/probe --destination ~/tmp/third-sat

Revision: 296
```

```
Date: 2011-10-06 01:16:57 +0200 (Thu, 06 Oct 2011)

./copy.py 1.1

------------------------------------------------------------------------------
* source      : ['/Users/clinares/tmp/results/seq-sat/probe']
* destination : /Users/clinares/tmp/third-sat
------------------------------------------------------------------------------
```

Now, `third-sat` contains the results of two planners:

```
$ dir ~/tmp/third-sat/
total 0
drwxr-xr-x  3 clinares  staff  102 Oct 11 16:09 lama-2011/
drwxr-xr-x  3 clinares  staff  102 Oct 11 15:59 probe/
```

for one particular domain:

```
$ dir ~/tmp/third-sat/lama-2011
total 0
drwxr-xr-x  23 clinares  staff  782 Oct 11 16:09 openstacks/
$ dir ~/tmp/third-sat/probe
total 0
drwxr-xr-x  22 clinares  staff  748 Oct 11 16:11 openstacks/
```

7.3.3 Accessing the new results tree directory

Finally, with a unique results tree directory containing the results of the planners of choice with regard to the selected domains of interest it is possible to access the results with the reporting tools in the package `IPCReport`.

In the following example, the results tree directory `third-sat` is directly accessed —i.e., no snapshot is generated. In this case we will require the reporting tools to show information on the number of problems solved and those where the planner failed:

```
 $ /report.py --directory ~/tmp/third-sat/ --variable numsolved oknumsolved numtimefails nummemfails
              --level planner

Revision: 297
Date: 2011-10-06 16:22:31 +0200 (Thu, 06 Oct 2011)

./report.py 1.2

  ------------------------------------------------------------------------------
  * directory      : /Users/clinares/tmp/third-sat
  * snapshot       :
  * name           : report
  * level          : planner
  * planner        : .*
  * domain         : .*
  * problem        : .*
  * variables      : ['numsolved', 'oknumsolved', 'numtimefails', 'nummemfails']
  * unroll         : False
  * sorting        : []
  * style          : table
  ------------------------------------------------------------------------------

  name: report
 +-----------+-----------+-------------+--------------+-------------+
```

```
|  planner   | numsolved | oknumsolved | numtimefails | nummemfails |
+------------+-----------+-------------+--------------+-------------+
| lama-2011 |     20    |      20     |      0       |      0      |
|   probe   |     4     |      4      |      16      |      0      |
+------------+-----------+-------------+--------------+-------------+
...

 created by IPCrun 1.2 (Revision: 295), Tue Oct 11 16:53:45 2011
```

From the preceding table it follows that while **lama-2011** solved all instances, **probe** only managed to solve four failing on time on all the rest. This is not surprising taking into account that these experiments were performed allowing **probe** to run only for 30 seconds per instance while **lama-2011** was given 30 minutes in the Seventh International Planning Competition.

7.4 Fourth practical case

In this practical case we want to know whether one sequential satisficing planner, **yahsp2**, performs better than its multithreaded version, **yahsp2-mt**, for solving a particular case: problem 001 of the domain openstacks of the sequential satisficing track.

The hypothesis set in this experiment are the following:

Null Hypothesis Both planners perform much alike and the mean time to generate solutions is the same.

Alternate Hypothesis One planner performs better than the other and tend to produce solutions faster.

The precise moments in time when both planners provided solutions can be found with the script `report.py` as follows [16]:

```
$ ./report.py --summary ./seq-sat.results.snapshot --variable timesols
              --domain openstacks --problem 001 --planner 'yahsp2'
```

Both series can be automatically retrieved by the script `test.py` using exactly the same parameters. However, it only accepts single-valued series so that the argument *--unroll* should be provided as well. `test.py` creates automatically two series from the results provided by `report.py`, one with the exact timings of all the solutions found by **yahsp2** and another for **yahsp2-mt**. This is done automatically because the leftmost column of the report obtained with the previous command (which is **planner**) contains the names of the series of interest: **yahsp2** and **yahsp2-mt**.

By simple inspection of the results of the previous report it seems that **yahsp2** tends to take longer to provide solutions yet, it produces 33 solutions, while **yahsp2-mt** seems to be faster but solving almost ten problems less [17]. The non-parametrical statistical test Mann Whitney U test can be used for this purpose. It just suffices to invoke it with the same parameters than above (and *--unroll* as already discussed) and *--test mw* to specify this particular test:

```
$ ./test.py --summary ./seq-sat.results.snapshot --variable timesols
            --domain openstacks --problem 001 --planner 'yahsp2' --unroll
            --test mw

   Revision
   Date
```

[16] The output is not shown because it exceeds the right margin of this document

[17] The reader is warned that all these solutions were found invalid by VAL so that the variables to use here will refer to the values of interest not validated by VAL.

```
./test.py 1.3

------------------------------------------------------------------------------
* snapshot       : ./seq-sat.results.snapshot
* tests          : ['Mann-Whitney U']
* name           : report
* level          : None
* planner        : yahsp2
* domain         : openstacks
* problem        : 001
* variable       : ['timesols']
* filter         : None
* matcher        : all
* noentry        : -1
* unroll         : True
* sorting        : []
* style          : table
------------------------------------------------------------------------------


name: report
+-----------+-------------------+-------------------+
|           |       yahsp2      |     yahsp2-mt     |
+-----------+-------------------+-------------------+
|   yahsp2  |        ---        | 2.43846017451e-05 |
| yahsp2-mt | 2.43846017451e-05 |        ---        |
+-----------+-------------------+-------------------+
Mann-Whitney U : Computes the Mann-Whitney rank test on two samples. It assesses whether one of tw
independent observations tends to have larger values than the other. This test corrects for ties a
by default uses a continuity correction. The reported p-value is for a one-sided hypothesis, to ge
the two-sided p-value multiply the returned p-value by 2.

created by IPCtest 1.3 (Revision: 312), Sun Jul 15 23:40:18 2012
```

The preceding table reports a *p*-value equal to 0.0000243 that one serie tends to have larger values than the other. Since this is the *p*-value of the one-sided hypothesis, the *Null Hypothesis* can be discarded even with a confidence level of α=0.001 (i.e., with a probability of 99.9% that the difference are real and not due to chance) and the *Alternate* or *Research Hypothesis* that **yahsp2-mt** is faster can be accepted.

For more information about this script see section *test.py* where another case with more options is discussed in depth.

Before finishing, this script acknowledges all the different styles provided by `report.py` with the directive `--style` so that the same tables can be shown in the markup languages html and wiki, Octave files and also in excel worksheets.

Note: To know more about their relative performance in these domain and others as well as performing comparisons with a large number of planners (including some programmed by you or your colleagues) you are invited to start using the software of the Seventh International Planning Competition.

New in version 1.1: three new variables are provided now: *lengths*, *maxlength* and *minlength*

CHAPTER

EIGHT

REPORTING VARIABLES

This section describes all the variables implemented in the reporting tools in the package `IPCReport`. When using the `report.py` script variables are specified with the directive `--variable`. However, there is a convenient shortcut to specify them as well `--vvar` where *var* shall be substituted by the corresponding variable.

There are two types of variables: *raw* and *elaborated*

8.1 Raw variables

Raw variables are those that read values directly from the terminal directories of the results directory —or, alternatively, from the summary that encompasses the same observation. For example, section *invokeplanner.py* shows all the files generated after running the sequential optimal planner **bjolp** with the first problem of the floortile domain. Information that can be directly retrieved by parsing the contents of these files is known as being *raw*.

In particular, the raw variables currently acknowledged by `report.py` (see *report.py*) are listed below:

	Raw variables	
logfile	vallogfile	track
subtrack	planner	domain
problem	timeout	membound
runtime	memend	memmax
numsols	valnumsols	oknumsols
timesols	oktimesols	timelabels
memlabels	timefirstsol	timelastsol
oktimefirstsol	oktimelastsol	solved
oksolved	plansoln	okplansoln
values	uppervalue	lowervalue
lengths	maxlength	minlength

logfile

Name of the logfile generated by `invokeplanner.py` when running the planner on the selected problem

vallogfile

Name of the logfile generated by the Automatic Validation Tool VAL when validating the results of a particular directory

track

Track this case belongs to

subtrack

Subtrack this case belongs to

planner

Name of the planner used in this particular experiment

domain

Name of the domain used in this particular experiment

problem

Problem identifier (a number in the range 000 .. 999) of this particular case

timeout

Timeout (in seconds) given to `invokeplanner.py` when making this experiment in the directive *--time*

membound

Memory limit (in bytes) given to `invokeplanner.py` when making this experiment in the directive *--memory*

runtime

Overall run time (in seconds) used by the planner for solving this particular problem. If the process started a number of threads and/or processes, this variable records the sum of all of them.

memend

Memory used by the planner (in bytes) at the end of the planning process. While most planners take memory incrementally, there are some who do not do it. In this case, this is not necessarily the memory peak used by the planner

memmax

Memory peak (in bytes) of the memory used by the planner during the whole planning process.

numsols

Number of plan solution files generated. While in the sequential optimal track this number shall be expected to be either 0 or 1, planners shall be expected to generate an arbitrary number of plan solution files.

valnumsols

Number of non-empty solution files generated. While the expected behaviour is that planners either generate plan solution files with some information or they do not, an additional check can be performed with this variable.

oknumsols

Number of plan solution files that have been successfully validated by the Automatic Validation Tool VAL

timesols

Time elapsed (in seconds) since the planner was started when different solutions were found. This variable can either be empty, contain a single value or an array of values.

oktimesols

Time elapsed (in seconds) since the planner was started when different valid solutions (validated by the Automatic Validation Tool VAL) were found. This variable can either be empty, contain a single value or an array of values.

timelabels

While the planner is running, the script `invokeplanner.py` samples the time and memory consumption of the planner at regular intervals. This variable records the time spent (in seconds) among successive samplings —currently, samplings are performed every 5 seconds.

memlabels

Memory consumption (in bytes) at each sampling as described in *timelabels*

timefirstsol

Time spent (in seconds) until the first solution has been found. This value is just the first one reported in *timesols* and might be the same than *timelastsol*

timelastsol

Time spent (in seconds) until the last solution has been found. This value is just the last one reported in *timesols* and might be the same than *timefirstsol*

oktimefirstsol

Time spent (in seconds) until the first valid solution (validated by the Automatic Validation Tool VAL) has been found. This value is just the first one reported in *oktimesols* and might be the same than *oktimelastsol*

oktimelastsol

Time spent (in seconds) until the last valid solution (validated by the Automatic Validation Tool VAL) has been found. This value is just the last one reported in *oktimesols* and might be the same than *oktimefirstsol*

solved

This is a boolean variable that takes the value `true` if the planner has generated a plan solution file and `false` otherwise. Whether the solution is correct or not is not taken into account in this variable.

oksolved

This variable that takes the value `true` if the planner has generated a valid solution and `false` otherwise. If no solution was validated, it takes the special value `?`

plansoln

Filenames of non-empty solution files generated by the planner for this particular case.

okplansoln

Filenames of non-empty valid solution files (validated by the Automatic Validation Tool VAL) generated by the planner for this particular case.

values

Values returned by the Automatic Validation Tool VAL for each plan that was found to be valid

uppervalue

Largest value returned by the Automatic Validation Tool VAL. While it is not necessarily true, this should be expected to be the first one of the values reported in *values*

lowervalue

Lowest value returned by the Automatic Validation Tool VAL. While it is not necessarily true, this should be expected to be the last one of the values reported in *values*

lengths

This variable is available since version 1.1. It returns a list with the number of operations in every plan validated by the Automatic Validation Tool VAL. This variable and the next two ones serve, for example, to compare the cost of the plans (which are stored in the variable *values*) with the number of actions in each plan.

maxlength

This variable is avaliable since version 1.1. It returns the largest step length returned by the Automatic Validation Tool VAL

minlength

This variable is avaliable since version 1.1. It returns the smallest step length returned by the Automatic Validation Tool VAL

8.2 Elaborated variables

A variable is said to be elaborated if it results from some computation from raw variables. In some cases, the elaborated variables might refer solely to raw variables of a particular case but most likely they result from the computation of raw variables from different cases simulatenously.

The elaborated variables currently acknowledged by `report.py` (see *report.py*) are summarized below:

	Elaborated variables	
sumruntime	minruntime	maxruntime
summemend	minmemend	maxmemend
summemmax	minmemmax	maxmemmax
sumnumsols	oksumnumsols	numprobs
numsolved	oknumsolved	numfails
timefails	memfails	unexfails
numtimefails	nummemfails	numunexfails
minvalue	maxvalue	

sumruntime

Sum of the overall runtime (raw variable *runtime*) of a selected subset of problems

minruntime

Minimum overall runtime (raw variable *runtime*) of a selected subset of problems

maxruntime

Maximum overall runtime (raw variable *runtime*) of a selected subset of problems

summemend

Sum of the values of the raw variable *memend* of a selected subset of problems

minmemend

Minimum value of the raw variable *memend* of a selected subset of problems

maxmemend

Maximum value of the raw variable *memend* of a selected subset of problems

summemmax

Sum of the values of the raw variable *memmax* of a selected subset of problems

minmemmax

Minimum value of the raw variable *memmax* of a selected subset of problems

maxmemmax

Maximum value of the raw variable *memmax* of a selected subset of problems

sumnumsols

Total number of solution files generated for a selected subset of planning tasks

oksumnumsols

Total number of solution files validated by the Automatic Validation Tool VAL

numprobs

Total number of problems in a particular selection

numsolved

Total number of problems for which at least one solution plan has been generated (which however might be invalid) among a selected subset of planning tasks

oknumsolved

Total number of problems that have been solved with valid solutions according to the Automatic Validation Tool VAL among a selected subset of planning tasks

numfails

Total number of problems not solved for a selected subset of planning tasks

timefails

Cases where the planner reached the time bound without ever generating a single solution file, for a selected subset of planning tasks

memfails

Cases where the planner reached the memory limit without ever generating a single solution file, for a selected subset of planning tasks

unexfails

Cases where the planner failed to generate at least a solution file and it terminated without reaching the time bound or the memory limit.

numtimefails

Number of *timefails* identified for a selected subset of planning tasks

nummemfails

Number of *memfails* identified for a selected subset of planning tasks

numunexfails

Number of *unexfails* identified for a selected subset of planning tasks

minvalue

Minimum cost returned by the Automatic Validation Tool VAL for a selected subset of planning tasks

maxvalue

Maximum cost returned by the Automatic Validation Tool VAL for a selected subset of planning tasks

CHAPTER

NINE

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

9.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

9.2 TERMS AND CONDITIONS

9.2.1 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

9.2.2 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free

programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

9.2.3 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

9.2.4 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

9.2.5 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

9.2.6 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

9.2.7 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

9.2.8 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

9.2.9 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9.2.10 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

9.2.11 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's

predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

9.2.12 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

9.2.13 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

9.2.14 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

9.2.15 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

9.2.16 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

9.2.17 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRO-

GRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9.2.18 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

CHAPTER

TEN

ACKNOWLEDGEMENTS

The Deterministic Part of the Seventh International Planning Competition would have not been the same without the collaboration of a long list of people. Therefore we wanted to include this section in appreciation of all the assistance provided by them.

First of all, we want to express our most sincere thanks to all the people who submitted a planner to the Competition.

Also, to all of you who suggested a domain to be included in the tracks, even if some were not accepted in the end: **Bharatranjan Kavuluri** suggested the *matchcellar* domain; **Frédéric Maris** sent us the *cooking* and *temporal machine shop* domains; **Jörg Hoffmann** and **Hootan Nakhost** devised the *nomystery* specification; **Ron Petrick** submitted the domain *crisp*; **Héctor Geffner** and **Nir Lipovetzky** coded the *visit-all* domain; **Amanda Coles** and **Andrew Coles** sent us the *market* domain; **Héctor Luis Palacios** prepared various conformant domains such as *1-dispose*, *dispose*, *grid*, *look-and-grab*, *push-to* and *classical*; **Bhaskara Marthi** is behind the *tidybot* domain; **Guy Shani** shall be acknowledged for sending the domains *colorballs doors*, *medicalPKS150*, *medicalPKS199*, *sliding-doors*, *unix* and *wumpus*. At last, but not least, **Tomás de la Rosa** worked in the *floortile* domain.

We do also want to explicitly express our gratitude to **Derek Long** for his assistance and support with the automated validation tool VAL.

Also, to **Daniel L. Kovacs** for making available through the wiki site of the Competition a couple of manuscripts with a formal specification of PDDL 3.1.

Besides, we do feel in debt with **Óscar Pérez**, **Jaime Pons**, **Roberto Fuentes** and, in general, to the Laboratory of the Computer Science Department of the University Carlos III de Madrid for their assistance in installing, configuring and maintaining the cluster.

Very importantly as well, to the IPC council for providing extensive comments and offering a lot of helpful suggestions. In particular, to **Malte Helmert** for inviting us to his university, his assistance with so much insight and all the material produced at the previous IPC.

To all the members of the Planning and Learning research group (PLG) of the University Carlos III de Madrid for their encouragement and, in so many cases, for being in charge of our daily duties to allow us to work in the Competition. In particular, to **Daniel Borrajo** for all the support.

A special mention shall be done in this document to **Malte Helmert** (again) for granting us access to his svn repository and all the code in general that he developed for the Sixth International Planning Competition. In fact, his name is preserved as well in a number of source files developed in the code for the Seventh International Planning Competition. Besides, **Sergio Jimenez** found a number of bugs and made a lot of nice suggestions during the design and development of all of these packages.

Finally, we have to acknowledge the sponsorship of Decide Soluciones, iActive, the University Carlos III de Madrid and ICAPS. The hardware platform used during the competition was funded by Spanish Science Ministry under project MICIIN TIN2008-06701-C03-03.

CHAPTER

ELEVEN

INDICES AND TABLES

- *genindex*
- *search*

INDEX

Symbols

B

C

D

E