Automating scientific experimentation

Universidad Complutense de Madrid

Carlos Linares López

Planning and Learning Group (PLG)
Computer Science Department
Universidad Carlos III de Madrid

January, 15, 2014

Motivation I

There's a lot involved in scientific experimentation![RU01, MS01, Joh02]

- Objectivity. The evaluation setting must be *objective* and it must be the same for all evaluated systems
- Exhaustiveness. The evaluation should support general conclusions
- Comparability. The evaluation should quantify progress with respect to previous evaluations.
- Reproducibility. The evaluation must be reproducible ensuring that the same results can be obtained when using the same experimental conditions
- Reliability. Evaluation systems should be error-free

Motivation II

Goal	Consideration
Objectivity	Representation language Evaluation metric Validation mechanism
Exhaustiveness	Difficulty measures Coverage analysis
Comparability	Baselines State-of-the-art systems
Reproducibility	Hardware details Software details Experimental setup Random behaviour
Reliability	Portable programming languages Testing

Table : Considerations

Motivation III

But we often just run our systems and extract a few metrics that are compared with current state of the art solvers —and we often do in a rush!
Still worse,

- Empirical evaluation tools are usually programmed *ad-hoc* $\longrightarrow$ lack of reusability
- It is more and more common to evaluate systems with regard to huge volumes of data (e.g., Competitions) which require specific means $\longrightarrow$ specific *ad-hoc* programs become more difficult

Solutions I

Domain-independent: usually based on the availability of large computing premises

- WINGS[http://www.wings-workflows.org/] is a semantic workflow system that assists scientists with the design of computational experiments
- STAREXEC[https://www.starexec.org/starexec/] provides queues for automating scientific experiments which can receive any solvers

Domain-dependent: other alternatives are domain-specific such as

- LAB[https://bitbucket.org/jendrikseipp/lab] automates all the experimentation with FAST-DOWNWARD which, in turn, includes various algorithms and heuristics

Solutions II

Another solution

To provide a domain-independent package suitable for smaller experiments:

- That integrates easily with larger computing premises (e.g., clusters)
- That can be easily extended for specific purposes

Outline

1. TESTBOT: a domain-independent package written in PYTHON
2. IPC-2014: a domain-specific implementation using TESTBOT

A bit of history ...

TESTBOT is strongly based on the software developed for the 2011 International Planning Competition[LJH13] ...
... which, in turn, is strongly based on the software developed for the 2008 International Planning Competition

Availability I

Download

TESTBOT is publicly available from `bitbucket.org`

```
hg clone ssh://hg@bitbucket.org/clinares/testbot
```

It is distributed under the terms of the GNU General Public License 3 or later (GPLv3+)

Availability II

Installation

TESTBOT is installed as:

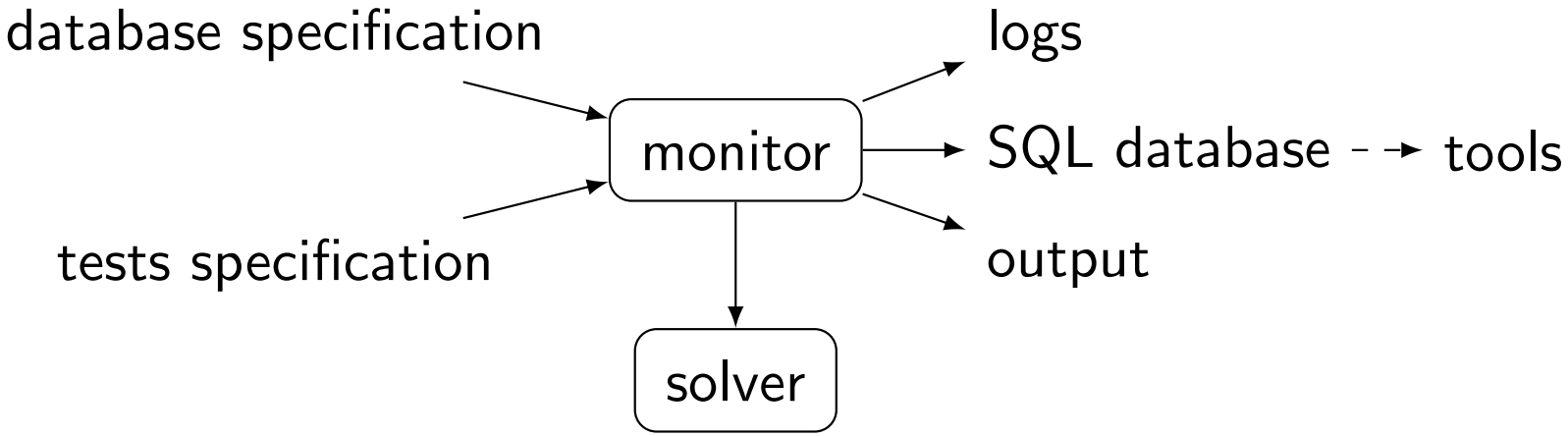
- AUTOBOT: a domain-independent package that provides various services for monitoring executables
- TESTBOT: a simple script that uses AUTOBOT

From a single-user domain-specific scenario . . .

Typical techniques include:

- SETRLIMIT over the process
- Additional scripts for extracting the relevant information from the formatted output

To a single-user domain-independent scenario . . .



The monitor:

- Starts/kills the solver
- Pings the solver with arbitrary precision and measures its consumption of computational resources
- Gather data and write it to a SQL database
- Along with other relevant information

TESTBOT I

- Test cases are distinguished by the arguments given to the *solver* —test specification file
- Data is explicitly declared in a database specification file
- Computational resources (*time* and *memory*) are explicitly declared
- Multiple *solvers* can be instantiated
- It implements logging services of various levels (`DEBUG`, `INFO`, `CRITICAL`, `ERROR`) on different streams (`FileHandler` and `StreamHandler`)

TESTBOT II

- TESTBOT monitors the execution of the underlying *solvers* with an arbitrary precision (examining `pgrp` and the `pid` of all processes)
- It includes summary of various data to any output directory:
  - Config information
  - Log information
  - Results
- It captures the *standard output* and *standard error* which can be compressed with `bzip2`
- Additionally, the *standard output* can be parsed to create variables which can be referenced in the database specification file

TESTBOT III

Additionally, it provides a lot of functionalities for developers of third-party software:

- Automatic access to the tests specification file and the database specification file
- which can be specified as files or verbatim strings
- Automated actions: `setUp`/`tearDown` (such as in `unittest`), `enter`/`windUp` and `prologue`/`epilogue`
- Transparent access to the values of all internal variables in testbot by using inheritance

A bit of history ...

IPC-2014 is the official software to be used in the 2014 International Planning Competition[1]

[1] http://helios.hud.ac.uk/scommv/IPC-14/

Availability

Availability

IPC-2014 is publicly available from `bitbucket.org`

```
hg clone ssh://hg@bitbucket.org/clinares/ipc-2014
```

It is distributed under the terms of the GNU General Public License 3 or later (GPLv3+)

Past of the IPC I

In 2008 there was an attempt to automate much of the burden involved in the automated evaluation of planning systems.

- Automating the experimentation
- Automatic generation of reports (including pdf format)

Past of the IPC II

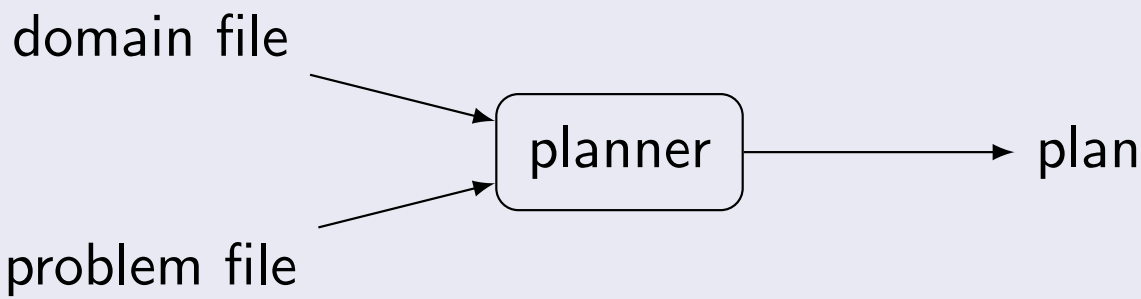
In 2011 most of these scripts were further generalized and some improvements were made in the *reporting* module:

- Automatic generation of reports in different formats (ASCII, XLS, OCTAVE, etc.)
- that automatically included any selection of 50+ different variables
- Sorted by any criteria
- Automatic ranking of solvers according to different *metrics* (such as *coverage*, *time*, *quality*, etc.)
- Pair-wise non-parametric statistical tests on any selection of planners according to any variable with outputs in various formats including `dot` (GRAPHVIZ)

Overall view I

Intuition · Deterministic part

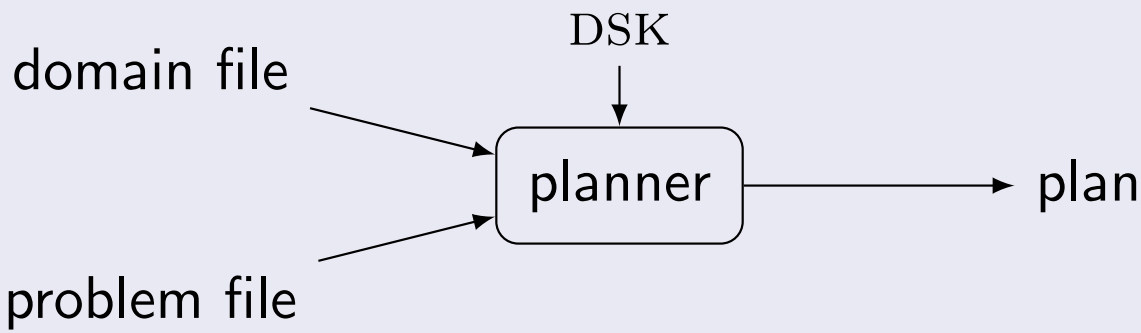
A planner is a domain-independent solver that receives a domain file (with the definition of *operators*) and a problem file (with the definition of *objects*, *start state* and *goals*) and outputs a plan $\pi\langle a_1, a_2, \ldots, a_n \rangle$ that fulfills the goals G from the initial state s



Overall view II

Intuition · Learning part

In the *Learning part* planners are allowed to exploit Domain-specific Knowledge (DSK) that is gathered in a preliminary step



Overall view III

Tracks

Still, there are tracks devoted to different models such as sequential optimal, satisficing, multi-core satisficing planning and also temporal satisficing planning
They induced different metrics, CPU/Wall-clock time control and validation processes

Functionality I

- Test specification file: all planners are invoked through a script PLAN which receives three parameters: *domain*, *problem* and prefix of the *plan solution file* —and additionally the DSK in the learning track
- Database specification file: it generates a unique SQL database that contains
  1. *Admin tables*: version, timeline, tests used, return code, etc.
  2. *Sys tables*: memory used, CPU/Wall-clock time, number of processes and threads (as a function of CPU/Wall-clock time)
  3. *Data tables*: parsed from the standard output of the solver
  4. *User tables*: computed by the code of the IPC-2014
     It mostly includes information about the (external) validation process such as *plan valid*, *plan cost*, *step length*, etc.

Functionality II

- Planners and domains are stored in a SVN repository and they are automatically checked out/compiled under demand
- Some (buggy) planners can generate huge output files so that *standard output* and *standard error* are automatically compressed
- It records a logfile (level `INFO`) with the output of the whole monitoring process
- It is not required to process the standard output

Integration in larger computer premises

It is known to integrate easily with CONDOR

```
nice_user = true
universe = vanilla
getenv = TRUE

Executable = script.sh
Output = test.$(Cluster).$(Process).out
log = test.$(Cluster).$(Process).log
error = test.$(Cluster).$(Process).err

request_memory = 7500
should_transfer_files = YES

transfer_input_files = ipc.ini, scripts
WhenToTransferOutput = ON_EXIT_OR_EVICT
transfer_output_files = exec_$(PROCESS)

arguments = spmas01 $(PROCESS)
queue 28
```

```
#!/bin/bash

# domain list
domains=("barman-opt11-strips$"
         "elevators-opt11-strips$" ...)

# Get the domain: $2 mod domains.length
num_domains=${#domains[@]}
num_dom=$(($2 % $num_domains))

# Define domain and exec folder
domain=${domains["$num_dom"]}
exec_folder=exec_"$2"

# Execution
mkdir "$exec_folder"
cd scripts/IPCData
./invokeplanner.py --path ... --repo ...
                   --domain ... --planner ...
```

In the 2014 IPC it will be used in a DES system

Minimum, maximum and average CPU time

```
[clinares@atlas madagascar-p]$ sqlite3 ./madagascar-p.db 'select MIN (cputime),
                                                          MAX(cputime), AVG(cputime)
                                                          from sys_time'
0.0|5.01|1.62550353837779
```

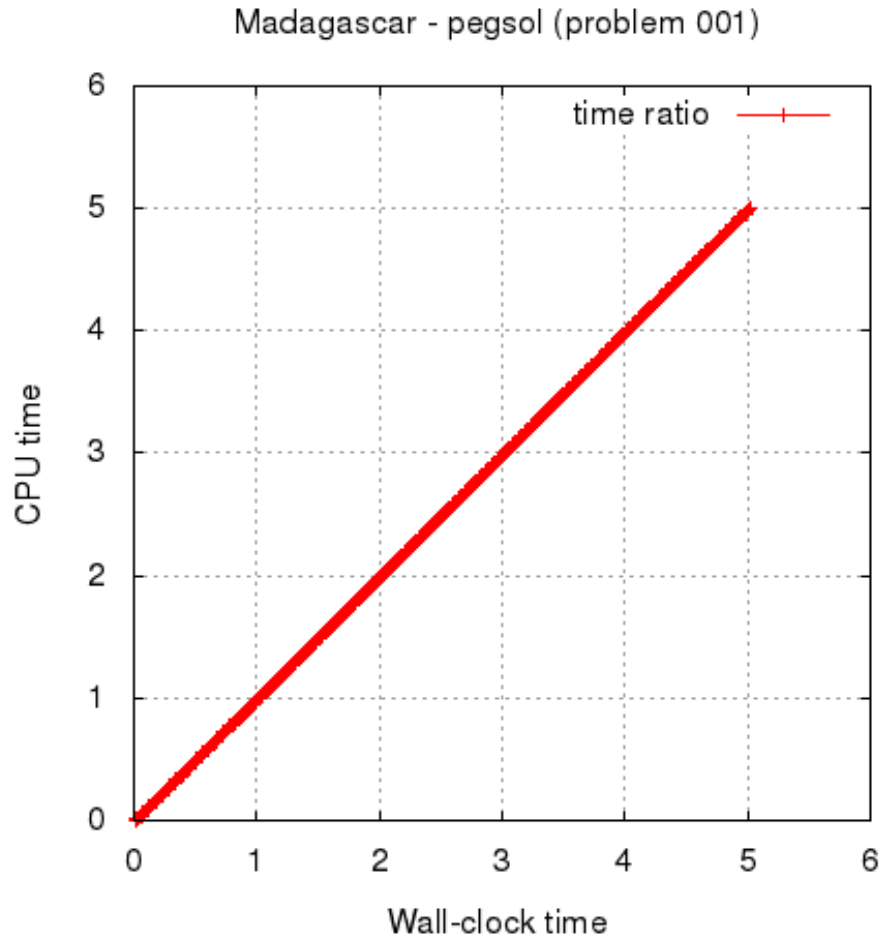
Solved instances

```
[clinares@atlas madagascar-p]$ sqlite3 ./madagascar-p.db 'select DISTINCT(problem)
                                                          from user_plan_processed'
000
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
```

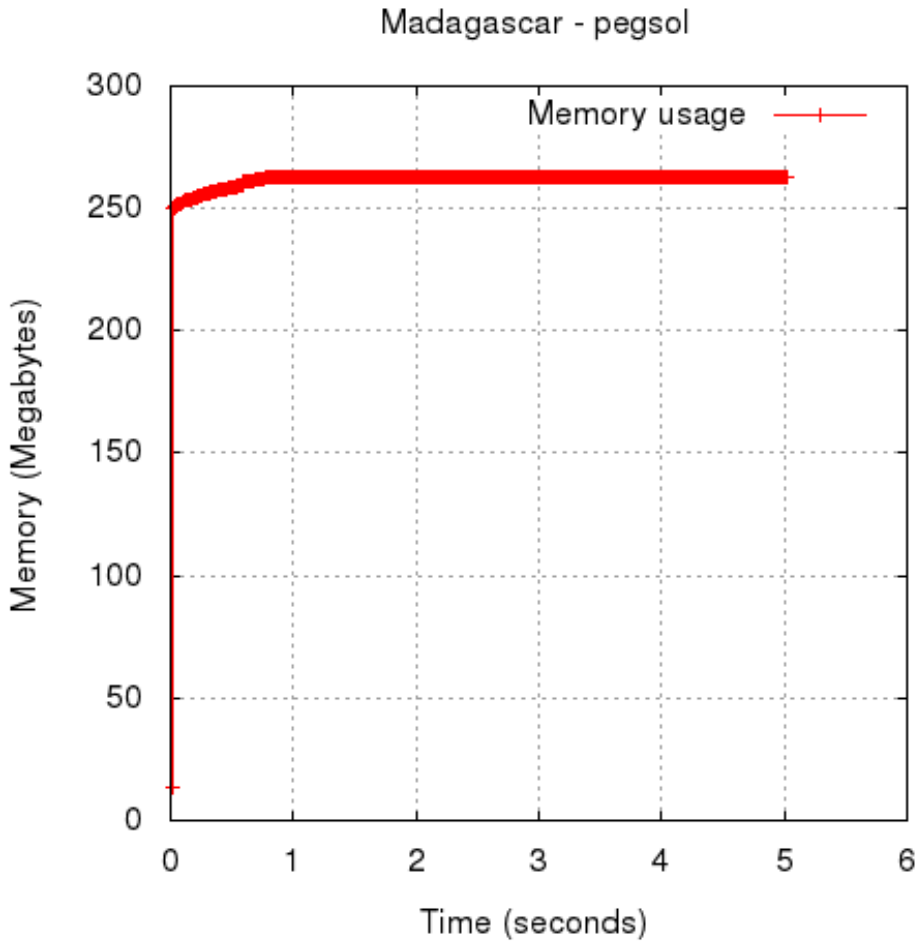
Valid solutions

```
[clinares@atlas madagascar-p]$ sqlite3 ./madagascar-p.db 'select DISTINCT(problem)
                                                          from user_plan_valid
                                                          where valid=1'
000
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
```

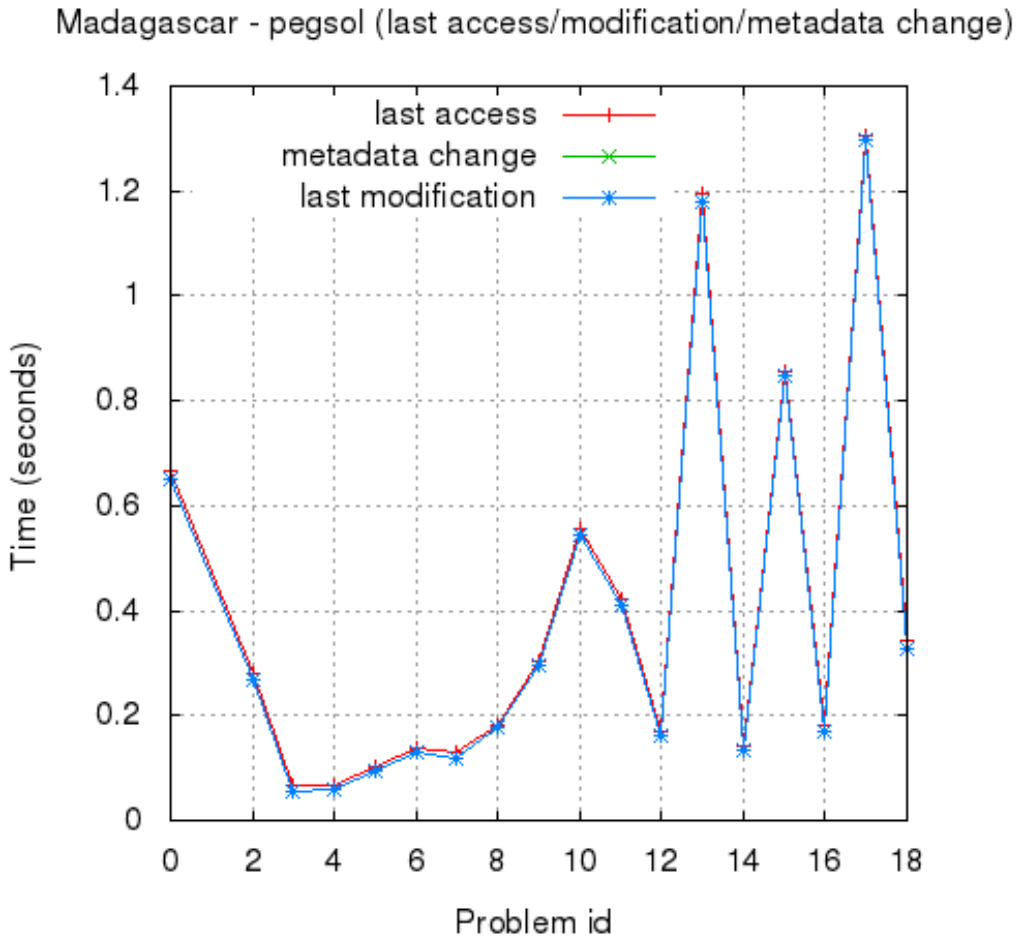
CPU time vs Wall-clock time (scatterplot)



Memory usage



Access/modification/metadata change



Plan length / step length

References I


David S. Johnson, *A theoretician's guide to the experimental analysis of algorithms*, Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges (2002), 215–250.


Carlos Linares López, Sergio Jiménez, and Malte Helmert, *Automating the evaluation of planning systems*, AI Communications **26** (2013), no. 4, 331–354.


Bernard M. E. Moret and Henry D. D. Shapiro, *Algorithms and experiments: The new (and old) methodology*, Journal of Universal Computer Science **7** (2001), no. 5, 434–446.


Ronald L. Rardin and Reha Uzsoy, *Experimental evaluation of heuristic optimization algorithms: A tutorial*, Journal of Heuristics **7** (2001), 261–304.