
perprof-py Documentation

Release 1.1.1

Raniere

Jun 25, 2017

Contents

1	Install	3
1.1	General	3
1.2	GNU/Linux Distribution	4
1.3	Mac OS X	4
1.4	Windows	5
2	Input File Format	7
2.1	YAML header	7
3	Old Input File Format	9
4	Command line flags	11
4.1	Store flags in a file	11
5	API for Developers	13
5.1	Entry point	13
5.2	File parse	13
5.3	Profile interface	14
5.4	Profiler using TikZ	14
5.5	Profiler using matplotlib	14
6	Indices and tables	15
	Python Module Index	17

perprof-py is a free/open source Python module for performance profiling (as described by [Dolan and Moré](#)) with output powered by

- [PGF/TikZ](#) + [PGFplots](#), for the case of writing articles in LaTeX;
- [matplotlib](#), for the case of PNG/JPG images to LibreOffice Writer or Microsoft Office Word; and
- [Bokeh](#), for the case of HTML pages with interactive exploration.

Contents:

CHAPTER 1

Install

Important: This package requires Python3 and isn't compatible with Python2.

Below you will find instructions to install perprof-py for any operation system that run Python and after that more detail instructions for some popular operation system.

General

Install perprof-py in a Python environment with pip is easy:

```
# pip install -r REQUIREMENTS
# python setup.py install
```

Note: You can [install it locally](#) if you want but to take advantage of bash completion we recomend you to install it globablly.

Note: For developers you can use [the develop mode](#) instead.

```
# python setup.py develop
```

To see a demo:

```
$ perprof --mp -o demo -f --demo
```

GNU/Linux Distribution

The general instructions probably work for you if you already using Python3 as default. Instructions for distributions that still using Python2 are found below.

Debian

```
# apt-get install python3 pip3
```

And follow the general instructions replacing `python` with `python3` and `pip` with `pip3`.

Ubuntu

```
# apt-get install python3 pip3
```

And follow the general instructions replacing `python` with `python3` and `pip` with `pip3`.

Fedora

```
# yum install python3 pip3
```

And follow the general instructions replacing `python` with `python3` and `pip` with `pip3`.

Arch

```
# pacman -S pip
```

And follow the general instructions.

Gentoo

```
# emerge pip
```

And follow the general instructions.

Mac OS X

Note: Comming soon.

Windows

Note: Comming soon.

CHAPTER 2

Input File Format

The current format start with a optional YAML header for metadata follow by the data as show in the template below:

```
---
<Metadata 01>: <Value 01>
<Metadata 02>: <Value 02>
---
<Problem Name 01> <Exit Flag 01> <Cost 01>
<Problem Name 02> <Exit Flag 02> <Cost 02>
<Problem Name 03> <Exit Flag 03> <Cost 03>
...
```

where

<Metadata XX> is the name of metadata field

<Value XX> is the value of the metadata field

<Problem Name XX> is the name of the problem

<Exit Flag XX> is c or d, meaning converged and diverged, respectively

<Cost XX> is the “cost” (e.g. time spent) to be used for the performance profile until solve the problem or give up.

Some examples of input file are provided at `perprof/examples` ([see it on GitHub](#)). To see the examples already

```
$ cd perprof/examples
$ ./make-examples.sh
```

This will generate 8 simple examples in the folder `perprof/examples/plots`.

YAML header

The YAML header store some useful metadata information and optionally some configurations.

Metadata

algname The name of the algorithmic/solver to be used in the plot. Default: File name.

col_dual The column corresponding to the dual feasibility at the solution. Default: 6

col_exit The column corresponding to the exit flag. Default: 2

col_fval The column corresponding to the objective function value at the solution. Default: 4

col_name The column corresponding to the problem names. Default: 1

col_primal The column corresponding to the primal feasibility at the solution. Default: 5

col_time The column corresponding to the time/cost spent on the problem. Default: 3

free_format Only check for mark of success. Default: False

maxtime The maximum time that a algorithmic/solver can run. Default: inf (i.e. not verified)

mintime The minimum time that a algorithmic/solver need to run. Default: 0

subset The name of the file to be used for the subset. Default: None

success List of strings to mark success. Default: 'c'

CHAPTER 3

Old Input File Format

Important: This is keep to backward compatibility.

The old format follow the template below:

```
#Name <Solver Name>
<Problem Name 01> <Exit Flag 01> <Cost 01>
<Problem Name 02> <Exit Flag 02> <Cost 02>
<Problem Name 03> <Exit Flag 03> <Cost 03>
...
```

where

<Solver Name> is the name of the solver to be used in the plot

<Problem Name XX> is the name of the problem

<Exit Flag XX> is c or d, meaning converged and diverged, respectively

<Cost XX> is the “cost” (e.g. time spent) to be used for the performance profile until solve the problem or give up.

Command line flags

To fully customize your needs, you may need to add a few flags for `perprof`. There are a lot of options, to see a list with all of them:

```
$ perprof --help
```

Some of the most important are

- `--semilog::` Use logarithmic scale for the x axis of the plot.
- `--success SUCCESS::` Set the values that are interpreted as success by `perprof`. Defaults to `c`.
- `--free-format::` Indicates that values that are not success should be accepted as failures.
- `-o NAME::` Sets the file name of the output.
- `-f::` Overwrite the output file, if it exists.

So, for instance, the call

```
$ perprof FILES -mp -o prof -f -semilog -success "optimal,conv" -free-format
```

calls `perprof` with Matplotlib, saves to file `prof.png`, overwrites if it exists, uses a semilog axis, set the success strings to `optimal` and `conv`, and indicates that every other exitflag string is a failure.

Store flags in a file

When calling `perprof` often, it is best to create a file storing your desired flags. You may then call this file with a `@` preceding the file name:

```
$ perprof @flagfile [more flags] FILE1 FILE2 [FILE3 ...]
```

The file `flagfile`, for our examples above, would be

```
-mp -o prof -f -semilog -success "optimal,conv" -free-format
```

Please note that the arguments in the file and in the command line are treated equally, so you can't add conflicting options.

Here you will find the API provide for developers.

Entry point

This is the main file for perprof

```
perprof.main.main()  
    This is the entry point when calling perprof.  
perprof.main.process_arguments(args)  
    Generates the dictionaries with options  
perprof.main.set_arguments(args)  
    Set all the arguments of perprof
```

File parse

Functions to parse the files

The files must be in the following format:

```
---  
<Metadata 01>: <Value 01>  
<Metadata 02>: <Value 02>  
---  
<Problem Name 01> <Exit Flag 01> <Cost 01>  
<Problem Name 02> <Exit Flag 02> <Cost 02>  
<Problem Name 03> <Exit Flag 03> <Cost 03>  
...
```

```
perprof.parse.parse_file(filename, parser_options)  
    Parse one file.
```

Parameters

- **filename** (*str*) – name of the file to be parser
- **options** (*dict*) – dictionary with the options: list subset: list with the name of the problems to use list success: list with strings to mark success int mintime: minimum time running the solver int maxtime: maximum time running the solver bool free_format: if False request that fail be mark with d

Returns performance profile data and name of the solver

Profile interface

The functions related with the perform (not the output).

class `perprof.prof.Pdata` (*parser_options*, *profiler_options*)
Store data for performance profile.

get_set_problems ()
Get the set of problems to use.

Returns list of problems

get_set_solvers ()
Get the set of solvers to use.

Returns list of solvers

plot ()
This should be implemented by a child of this class.

scale ()
Scale time.

set_percent_problems_solved_by_time ()
Set the percent of problems solved by time.

`perprof.prof.load_data` (*parser_options*)
Load the data.

Parameters **parser_options** (*dict*) – the configuration dictionary

Profiler using TikZ

This handle the plot using tikz.

class `perprof.tikz.Profiler` (*parser_options*, *profiler_options*)
The profiler using TikZ.

plot ()
Create the performance profile using TikZ/PgfPlots.

Profiler using matplotlib

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

p

- `perprof.main`, [13](#)
- `perprof.parse`, [13](#)
- `perprof.prof`, [14](#)
- `perprof.tikz`, [14](#)

G

`get_set_problems()` (perprof.prof.Pdata method), 14
`get_set_solvers()` (perprof.prof.Pdata method), 14

L

`load_data()` (in module perprof.prof), 14

M

`main()` (in module perprof.main), 13

P

`parse_file()` (in module perprof.parse), 13
`Pdata` (class in perprof.prof), 14
`perprof.main` (module), 13
`perprof.parse` (module), 13
`perprof.prof` (module), 14
`perprof.tikz` (module), 14
`plot()` (perprof.prof.Pdata method), 14
`plot()` (perprof.tikz.Profiler method), 14
`process_arguments()` (in module perprof.main), 13
`Profiler` (class in perprof.tikz), 14

S

`scale()` (perprof.prof.Pdata method), 14
`set_arguments()` (in module perprof.main), 13
`set_percent_problems_solved_by_time()` (perprof.prof.Pdata method), 14