Perprof-py: a Python package for performance profile of mathematical optimization software*

Abel Soares Siqueira[1], Raniere Gaia Costa da Silva[2] and Luiz-Rafael Santos[3]

1 - Department of Mathematics, Federal University of Paraná, Brazil - abelsiqueira@ufpr.br
2 - Institute of Mathematics, Statistics and Scientific Computing, University of Campinas, Brazil - raniere@ime.unicamp.br
3 - Department of Exact Sciences, Federal University of Santa Catarina, Brazil - l.r.santos@ufsc.br

Abstract

A very important area of research in the field of Mathematical Optimization is the benchmarking of optimization packages to compare solvers. During benchmarking, one usually collects a large amount of information like CPU time, number of functions evaluations, number of iterations, and much more. This information, if presented as tables, can be difficult to analyze and compare due to large amount of data. Therefore tools to better process and understand optimization benchmark data have been developed. One of the most widespread tools is the *Performance Profile* graphics proposed by Dolan and Moré [2]. In this context, this paper describes perprof-py, a free/open source software that creates Performance Profile graphics. This software produces graphics in PDF using LaTeX with PGF/TikZ [22] and PGFPLOTS [4] packages, in PNG using matplotlib [9], and in HTML using Bokeh [1]. Perprof-py can also be easily extended to be used with other plot libraries. It is implemented in Python 3 with support for internationalization, and is under the General Public License Version 3 (GPLv3).

1 Overview

Introduction

When creating a piece of software, the measurement of a set of information of interest regarding performance — for instance: CPU time, number of functions evaluated, number of iterations, memory usage, accuracy, or others — is common. *Benchmarking* is the process of measuring the performance information of one piece of software relative to similar software.

This is necessary when developing programs since it helps uncover software deficiencies and usually leads to general software improvements [3, 10, 11].

Furthermore, given a set of software that solve the same problem, one could compare them in order to choose the best one, or verify how their own software can be improved. To address this, Dolan and Moré [2] developed a tool to visualize optimization solvers benchmarks: the *performance profile*.

Formally, a performance profile allows one to evaluate and compare the performance of a set $\mathcal{S}$ of solvers on a given test set $\mathcal{P}$, with respect to a chosen evaluation parameter, which will be referenced as *cost*. It is presented as a graphic that shows the cumulative distribution function of different solvers performances, according to the chosen cost metric. Note that the cost metric must be positive.

This method is mostly used for nonlinear optimization solvers, however, it is possible to extend it to other software comparison. For instance, some authors have used it in the context of algorithms for matrix functions [5, 6, 7, 8, 12, 13, 14]. Notice that, in some cases, a specialized test can be more significant than the performance profile with a specific cost. For derivative-free optimization, for instance, Moré and Wild [15] define a *data profile*, using the number of function evaluations as the metric cost, nevertheless in a different way of performance profile definition.

For each problem $p \in \mathcal{P}$ and solver $s \in \mathcal{S}$, let $t_{p,s}$ be the cost required to solve problem p by solver s and

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}$$

*Software Metapaper

be the performance ratio of solver s for the problem p when compared with the best performance by any solver on this problem. As a convention, we set $r_{p,s}$ to a large value, say $r_{\max}$, if the solver s does not solve the problem p.

The probability of a solver $s \in \mathcal{S}$ to solve one problem within a factor $\tau \in \mathbb{R}$ of the best performance ratio is the function

$$\rho_s(\tau) = \frac{|\{p \in \mathcal{P} : r_{p,s} \leq \tau\}|}{|\mathcal{P}|}.$$

For a given τ, the best solver is the one with the highest value for $\rho_s(\tau)$, that is, the one with the highest probability to solve the problem. The value $\rho_s(\tau)$ represents the percentage of problems solved by algorithm s with a cost at most τ times worse than the best algorithm. $\rho_s(1)$ is the percentage of problems solved as fast as the fastest algorithm, which gives the efficiency of solver s. On the other hand

$$\lim_{\tau \to r_{\max}} \rho_s(\tau)$$

is the total percentage of problems solved by solver s, or in other words, the robustness of solver s.

Motivation

To facilitate the reproduction of data set analysis, such as benchmarking of solvers analysis provided by Dolan and Moré [2]'s performance profile, an open source tool that handles the production of performance profile plots should be available.

Performance profile has been, over the years, the most used benchmark comparison tool used in optimization. Nevertheless, the production of such analysis is sometimes a dull task, that can lead a researcher to waste a lot of time and effort that should have been spent in developing the solver itself.

There are other implementations to generate performance profiles, some of them being reasonable well-known, such as a MatLab script from the same group that created performance profile [17], and a module written by Michael Friedlander inside NLPy [18].

Some, perhaps unaware of these implementations or trying to avoid proprietary solutions, implemented their own solutions and then made them available. Solutions such as a Python function perfprof from Relton [20] and a Julia module `perfprof.jl` from Zhang [25], both language dependent. A thorough search would possibly reveal many others. However, there are features that some users need that these software have not implemented.

This paper describes a straightforward open source tool that allows one to create performance profile pictures in a fast and easy manner called perprof-py. In addition, this tool allows LaTeX users, a group which includes almost all of the optimization community, to generate performance profile plots as LaTeX code that will be processed later with the rest of their document or standalone PDF when needed.

With these two main goals in mind, perprof-py was developed and implemented in Python 3 with internationalization features and direct LaTeX integration.

Implementation and architecture

Perprof-py was implemented as a Python 3 package and organized to allow addition of new backends. Core files are

- `perprof/prof.py` that defines a class `Pdata` that need to be extend for every backend;
- `perprof/parse.py` that has the parser for the input files; and
- `perprof/main.py` that has the command line interface.

The incompatibility of perprof-py with Python 2 was due (i) to the fact that unicode processing with Python 2 can be a nightmare, and (ii) to the authors' desire to push Python 3 forward.

Users have a command line interface to use out of the box, however one can also use the package in their own software.

Implementation is very straightforward. In fact, the algorithm:

1. parses the options passed as arguments, creating a structure with all information;
2. parses and process input files, using the performance function definition to create data to be plotted;
3. uses the chosen backend to plot data.

Input

For each solver to be compared in benchmarking, one must write a file in the following manner:

```
---
YAML information
---
Problem01 exit01 time01
Problem02 exit02 time02
```

YAML[19, 24] information is a list of keywords and values used to set the name of the solver and some flags for perprof-py. A legacy option remains, in which the user can instead put only the solver name using

```
#Name SOLVERNAME
Problem01 exit01 time01
Problem02 exit02 time02
```

nevertheless some users may like to add more options.

Each line of data has at least 3 columns. Columns' meanings, in the default order, are:

- Problem's name;
- Exit flag;
- Cost measure – for instance, elapsed time.

Default exit flag is `c` or `d` on the exit flag column, meaning convergence or divergence, respectively.

One of the perprof-py solver examples uses the following YAML information

```
algname: Alpha
success: converged
free_format: True
```

which means that the name appearing on the profile will be `Alpha`; that `converged` is the word that means convergence, and that every other exit flag word means divergence. These options were set from `algname`, `success`, and `free_format` options, respectively.

A user can, optionally, add more columns to add information. They can verify, for instance, that the optimality conditions are satisfied for each problem. Also, using either YAML or command line options, a user can change each columns' meaning. Note that these options are not enabled by default. The user should consult the help and documentation files to see how to enable them.

Parsing process and output

To use perprof-py, one needs to issue a command of the type

```
$ perprof OPTIONS BACKEND FILES
```

where

- FILES are the input files described in the previous section. At least two files input are required;
- BACKEND is one of the options `--tikz`, `--mp`, `--bokeh` or `--raw`, which represents whether the user wants to use TikZ/PGFPLOTS, matplotlib, Bokeh, or simply printing the performance ratios, respectively;
- OPTIONS are varied arguments that can be passed to perprof-py to customize the graphics or modify the performance functions. Some noteworthy options are
  - `--semilog`: the natural logarithmic scale is used on the abscissa axis;
  - `--success STR`: `STR` is a comma separated string of keys that was considered *success* by the solver;
  - `--black-and-white`: perprof-py creates the plots using only line styles and it colors them in black;

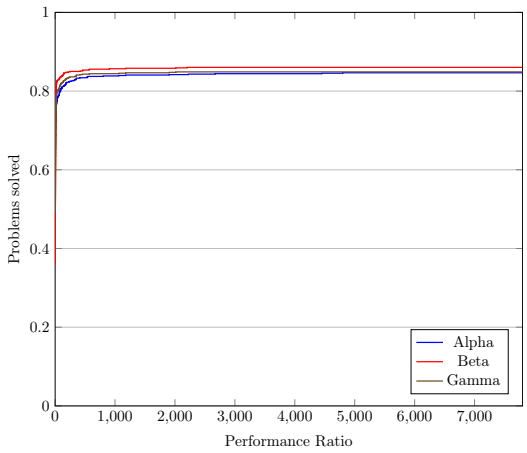

Figure 1: Example of performance profile with default options.

– `--subset FILE`: perprof-py considers only the subset problems listed in `FILE`, while creating the performance functions.

In order to demonstrate such OPTIONS, Figures 1-4 show some examples of performance profile graphics. Figure 1 shows the performance profile graphic with default options. Note that the lines are clumped due to the maximum time allowed in the solver. Figure 2 shows the performance profile using semilog option, which plots the graphic using a log scale on the abscissa. Figure 3 shows the performance profile using also the black and white option,

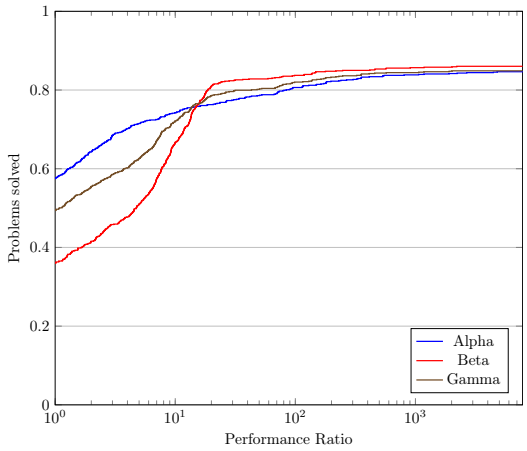

Figure 2: Example of performance profile with semilog option.

which gives a printer-friendly graphic. Figure 4 shows the performance profile using subset and semilog options. In this case, we selected around 120 problems, put their names in a file, and passed the file with the option. This limits the comparison to only those files.

Quality control

Perprof-py code is tested using unit tests that verify if incorrect input information is captured. These tests are run automatically on Travis CI [23], for Python 3.3 and 3.4. In addition, a script is run to generate several performance profile graphics. This script is also run on Travis CI, though the evaluation that perprof-py outputs the desired graphics can only be done locally.

This script uses artificial solver information accessible using `--demo` as argument in the perprof-py call. For instance, to test that the TikZ installation is successful, one can run

```
perprof-py --demo -o tmp --tikz
```

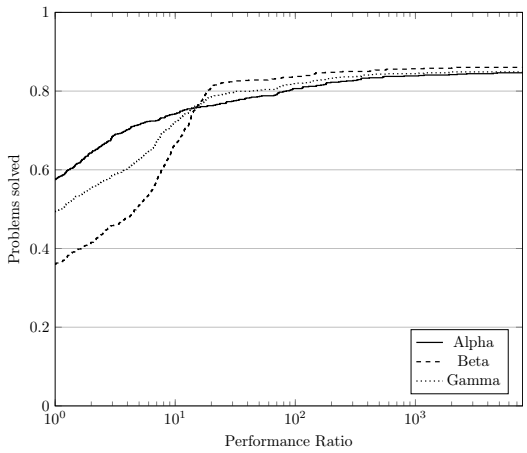

Figure 3: Example of performance profile with semilog and black and white options.

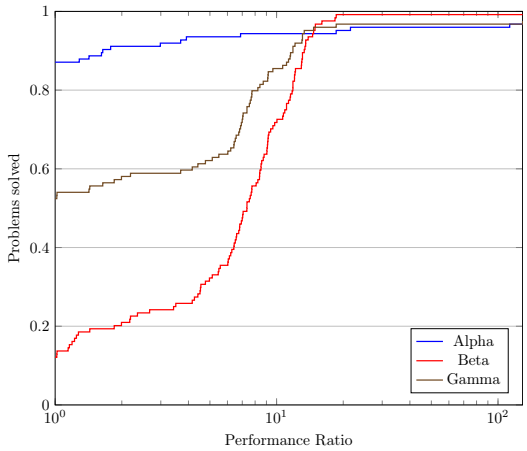

Figure 4: Example of performance profile with semilog and subset options.

If everything is correct, this will generate a file `tmp.pdf` with an example performance profile made using LaTeX and compiled to a standalone PDF.

One can run the testing script by entering the folder `perprof/examples` relative to the package folder, and running

```
./make-examples.sh
```

Folder `plots` will contain the outputs in formats PNG and PDF.

2 Availability

Operating system

Perprof-py is developed and actively tested on Unix platforms. The authors did not test it on Windows.

Programming language

The project was built entirely in Python 3.

Additional system requirements

No additional hardware requirements are necessary.

Dependencies

Perprof-py depends on the Python packages `matplotlib`, `pyyaml` and `bokeh`. In addition, if a user wants the PDF image from the LaTeX version, it also requires `pdflatex`.

Archive

Name:

perprof-py v1.1.1

Identifier:

`http://dx.doi.org/10.5281/zenodo.30031`

Licence:

GPL (General Public License) Version 3

Date published:

31/08/15

Publisher:

Abel Soares Siqueira

Date published:

31/08/15

Code Repository

Name:

GitHub

Identifier:

`https://github.com/ufpr-opt/perprof-py`

Licence:

GPL (General Public License) Version 3

Date published:

31/08/15

Language

Perprof-py was entirely developed in English, however there is support for other languages in the code. Currently, in addition to English, Brazilian Portuguese is the only other language implemented.

3 Reuse potential

The implementation of perfprof-py is separated in a way that facilitates the creation of a new backend. The class `Pdata` is defined to store the parsed data ($\mathcal{P}$, $\mathcal{S}$, $t_{s,p}$, etc.) and methods are defined to create the profile data $r_{p,s}$. Backends are classes that extend `Pdata` defining a method `plot` which creates the expected figure. One should have little difficulty creating their own backend, specially if one uses a perprof-py backend as a starting point. However, if one wants to change the profile data definition — in order to implement a data profile (see [16]) —, one would have to modify one or more methods in `Pdata` directly or re-implement the backends.

The parser opens the input files and creates the information for `Pdata`. Replacing this parser — to use with perprof-py backends — would not be an easy task since the correct output format should be created. Nevertheless, extending it with additional options would be simple enough.

The entry point `perprof-py` essentially collects the options from the command line and calls the specific backend profiler. This can be completely bypassed by calling the backend directly. This allows one to create a performance profile from another python application. In particular, a possibility is the creation of a graphical user interface (GUI) or a web server application. Perprof-py modularity allows whoever desires to construct this interface to focus entirely on obtaining the options from the user and passing it to the backend.

Whether one is planning on expanding some of perprof-py functionalities or creating any new backend or interface, one can contact the authors using the project page on GitHub [21].

Acknowledgments

The authors would like to thank FAPESP[1] and CNPq[2] for the partial support given to this project and their colleagues from LPOO and IMECC/UNICAMP. In addition, the authors are grateful for the valuable insights and suggestions given by Miles Lubin, by the editor-in-chief of JORS Neil Hong, and by the three anonymous reviewers, which improved `perprof-py` and this paper.

References

[1] *Bokeh*. URL: `http://bokeh.pydata.org/` (visited on 05/30/2015).

[2] E. D. Dolan and J. J. Moré. "Benchmarking optimization software with performance profiles". In: *Mathematical Programming* 91.2 (2002), pp. 201–213. DOI: `10.1007/s101070100263`.

[3] E. D. Dolan, J. J. Moré, and T. S. Munson. "Optimality Measures for Performance Profiles". In: *SIAM Journal on Optimization* 16.3 (Jan. 2006), pp. 891–909.

[4] C. Feuersänger. *Manual for Package PGFPLOTS*. Version 1.9. 2013.

[5] N. J. Higham. "The scaling and squaring method for the matrix exponential revisited". In: *SIAM Journal on Matrix Analysis and Applications* 26.4 (2005), pp. 1179–1193.

[6] N. J. Higham. "The scaling and squaring method for the matrix exponential revisited". In: *SIAM Review* 51.4 (2009), pp. 747–764.

[7] N. J. Higham and L. Lin. "A Schur-Pade algorithm for fractional powers of a matrix". In: *SIAM Journal on Matrix Analysis and Applications* 32.3 (2011), pp. 1056–1078.

[8] N. J. Higham and L. Lin. "An improved Schur-Pade algorithm for fractional powers of a matrix and their Frechet derivatives". In: *SIAM Journal on Matrix Analysis and Applications* 34.3 (2013), pp. 1341–1360.

[9] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[10] H. D. Mittelmann. "Benchmarking interior point LP/QP solvers". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 655–670.

[11] H. Mittelmann. *Benchmark for optimization software*. URL: `http://plato.la.asu.edu/bench.html` (visited on 08/21/2015).

[12] A. H. Al-Mohy and N. J. Higham. "A new scaling and squaring algorithm for the matrix exponential". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2009), pp. 970–989.

[1] Grants 2008/09685-8 and 2009/17273-4.
[2] Grant 501763/2013-9.

[13] A. H. Al-Mohy and N. J. Higham. "Computing the action of the matrix exponential, with an application to exponential integrators". In: *SIAM Journal on Scientific Computing* 33.2 (2011), pp. 488–511.

[14] A. H. Al-Mohy and N. J. Higham. "Improved inverse scaling and squaring algorithms for the matrix logarithm". In: *SIAM Journal on Scientific Computing* 34.4 (2012), pp. C153–C169.

[15] J. J. Moré and S. M. Wild. "Benchmarking Derivate-Free Optimization Algorithms". In: *SIAM Journal on Optimization* 20 (2009), pp. 172–191.

[16] J. J. Moré and S. M. Wild. "Benchmarking Derivative-Free Optimization Algorithms". In: *SIAM Journal on Optimization* 20.1 (2009), pp. 172–191. DOI: 10.1137/080724083.

[17] J. Moré, A. Bondarenko, D. Bortz, E. Dolan, M. Merritt, and T. Munson. *COPS: Large-Scale Optimization Problems*. URL: http://www.mcs.anl.gov/~more/cops/ (visited on 08/21/2015).

[18] D. Orban and M. Friedlander. *NLPy*. URL: https://github.com/dpo/nlpy (visited on 05/30/2015).

[19] *PyYAML*. URL: http://pyyaml.org/ (visited on 08/25/2015).

[20] S. Relton. *perfprof*. URL: https://github.com/sdrelton/perfprof (visited on 08/26/2015).

[21] A. S. Siqueira, R. G. C. da Silva, and L.-R. Santos. *Perprof-py*. URL: https://github.com/ufpr-opt/perprof-py/ (visited on 05/30/2015).

[22] T. Tantau. *The TikZ and PGF Packages. Manual for version 2.10-cvs*. 2012.

[23] *Travis CI*. URL: http://travis-ci.org (visited on 05/30/2015).

[24] *YAML Ain't Markup Language*. URL: http://yaml.org/ (visited on 08/25/2015).

[25] W. Zhang. *PerfPlot.jl*. URL: https://github.com/weijianzhang/PerfPlot.jl (visited on 08/26/2015).