Fachada, N et al 2016 PerfAndPubTools – Tools for Software Performance Analysis and Publishing of Results. *Journal of Open Research Software,* 4: e18, DOI: http://dx.doi.org/10.5334/jors.115

SOFTWARE METAPAPER

PerfAndPubTools – Tools for Software Performance Analysis and Publishing of Results

Nuno Fachada[1], Vitor V. Lopes[2], Rui C. Martins[3] and Agostinho C. Rosa[1]

[1] Institute for Systems and Robotics, LARSyS, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
nfachada@laseeb.org

[2] Universidad de las Fuerzas Armadas-ESPE, Sangolquí, Ecuador

[3] Life and Health Sciences Research Institute, School of Health Sciences, University of Minho, Braga, Portugal

Corresponding author: Nuno Fachada

PerfAndPubTools consists of a set of MATLAB/Octave functions for the post-processing and analysis of software performance benchmark data and producing associated publication quality materials.

Keywords: Profiling; Benchmarking; Performance analysis; Publishing tools
Funding statement: This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) projects UID/EEA/50009/2013 and UID/MAT/04561/2013, and partially funded with grant SFRH/BD/48310/2008, also from FCT. The author Vitor V. Lopes acknowledges the financial support from the Prometeo project of SENESCYT (Ecuador).

(1) Overview

Introduction

PerfAndPubTools consists of a set of MATLAB [1], GNU Octave-compatible [2], functions for the post-processing and analysis of software performance benchmark data and producing associated publication quality materials. More specifically, the functions bundled with PerfAndPubTools allow to:

1. Batch process files containing benchmarking data of computer programs, one file per run.
2. Determine the mean and standard deviation of benchmarking experiments with several runs.
3. Organize the benchmark statistics by program *implementation* and program *setup*.
4. Output scalability and speedup data, optionally generating associated figures.
5. Create publication ready benchmark comparison tables in LATEX.

These tools were originally developed to assess the performance of serial and parallel implementations of the PPHPC simulation model [3], as well as for producing some of the associated publication quality materials. However, the tools can be used with any computational benchmark experiment.

Implementation and architecture

Performance analysis in PerfAndPubTools takes place at two levels: *implementation* and *setup*. The *implementation* level is meant to be associated with specific software implementations for performing a given task, for example a particular sorting algorithm or a simulation model realized in a certain programming language. Within the context of each implementation, the software can be executed under different *setups*. These can be different computational sizes (e.g., vector lengths in a sorting algorithm) or distinct execution parameters (e.g., number of threads used).

PerfAndPubTools is implemented in a layered architecture using a procedural programming approach, as shown in **Figure 1**. From lowest to highest-level of functionality, the functions represented in this Figure have the following roles:

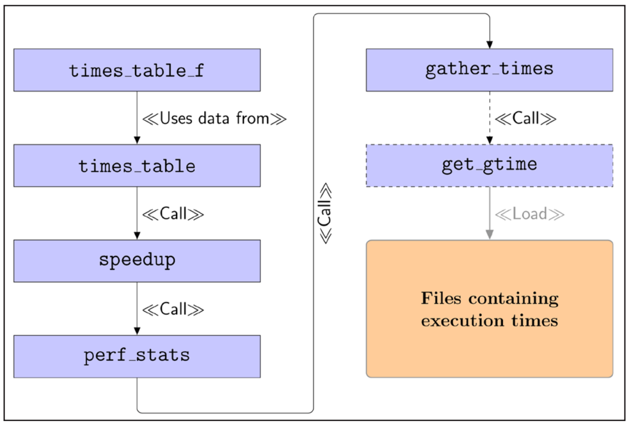

Figure 1: PerfAndPubTools architecture. Blocks in typewriter font represent functions. Dashed blocks represent directly replaceable functions.

get_gtime Given a file containing the default output of the GNU time [4] command, this function extracts the user, system and elapsed times in seconds, as well as the percentage of CPU usage.

gather_times Loads execution times from files in a given folder. This function uses get_gtime by default, but can be configured to use another function to load individual benchmark files with a different format.

perfstats Determines mean times and respective standard deviations of a computational experiment, optionally plotting a scalability graph if different *setups* correspond to different computational work sizes.

speedup Determines the average, maximum and minimum speedups against one or more reference *implementations* across a number of *setups*. Can optionally generate a bar plot displaying the various speedups.

times_table Returns a matrix with useful contents for using in tables for publication, namely times (in seconds), absolute standard deviations (seconds), relative standard deviations, and speedups against one or more reference *implementations*.

times_table_f Returns a table with performance analysis results formatted in plain text or in LATEX (the latter requires the siunitx [5], multirow [6] and booktabs [7] packages).

Although the perfstats and speedup functions optionally create plots, these are mainly intended to provide visual feedback on the performance analysis being undertaken. Those needing more control over the final figures can customize the generated plots via the returned figure handles or create custom plots using the data provided by perfstats and speedup. Either way, MATLAB/Octave plots can be used directly in publications, or converted to LATEX using the excellent matlab2tikz script [8], as exemplified in the PerfAndPubTools user manual.

An example: comparing sorting algorithms

Experimental setup: The performance of four sorting algorithms, implemented in C [9], is compared using PerfAndPubTools. The algorithms, Bubble sort, Selection sort, Merge sort and Quicksort [10], are used to sort random integer vectors of sizes 1 x 10^5, 2 x 10^5, 3 x 10^5 and 4 x 10^5. For each size, individual algorithms are executed ten times. Each run is benchmarked with GNU time, the output of which is redirected to a file with the following identifiers in its name: algorithm employed, run number and vector size.

In this context, a sorting algorithm is an *implementation*, and each vector size is a *setup*.

Defining implementation specs: Implementation specs are the basic objects accepted by the perfstats, speedup and times_table functions. An implementation spec defines one or more setups for a single implementation. A setup is defined by the following fields: a) sname, the name of the setup; b) folder, the folder where to load benchmark files[1] from; c) files, the specific files to load (using wildcards); and, d) csize, an optional computational size for plotting purposes. Multiple implementation specs can be defined, allowing PerfAndPubTools to compare multiple implementations across different setups.

In the following paragraphs, implementations specs stipulating all the setups (i.e., vector sizes) for the Bubble sort, Selection sort, Merge sort and Quicksort algorithms are represented by the bs, ss, ms and qs variables, respectively. For example, the Bubble sort implementation spec, bs, can be specified as follows:

```
datadir = 'path/to/files';

bs1e5 = struct('sname', '1e5', ...
   'folder', datadir, ...
   'files', '*_bubble_100000_*.txt', ...
   'csize' ,1e5);
bs2e5 = struct('sname', '2e5', ...
   'folder', datadir, ...
   'files', '*_bubble_200000_*.txt', ...
   'csize' ,2e5);
bs3e5 = struct('sname', '3e5', ...
   'folder', datadir, ...
   'files', '*_bubble_300000_*.txt', ...
   'csize', 3e5);
bs4e5 = struct('sname', '4e5', ...
   'folder', datadir, ...
   'files', '*_bubble_400000_*.txt', ...
   'csize', 4e5);

bs = {bs1e5, bs2e5, bs3e5, bs4e5};
```

Implementation specs for the remaining algorithms are defined in a similar fashion. Note that all implementations specs must have the same number of setups, and corresponding setups should have the same sname. Additionally, plotting with perfstats requires that the computational size, csize, is defined and has the same value for corresponding setups in different implementations specs.

Algorithm scalability: The perfstats function determines mean times and standard deviations of individual setups for each implementation. If the various setups correspond to different computational work sizes, perfstats can optionally plot a scalability graph. The following instruction performs this task for the experimental setup under discussion:

```
[m, s] = perfstats(3, 'Bubble', bs, ...
   'Selection', ss, 'Merge', ms, ...
   'Quick', qs);
```

The contents of the returned variables, m and s, are as follows:

```
m=
   36.0040    144.8210    325.1730    577.8600
    9.5270     38.0500     88.5130    153.6560
    0.0200      0.0410      0.0600      0.0850
    0.0100      0.0200      0.0300      0.0510
s =
    0.8873      2.9223      6.1874      6.3846
    0.0690      0.2829      3.6976      3.0600
    0.0000      0.0032      0.0000      0.0127
    0.0000      0.0000      0.0000      0.0032
```

The m variable represents mean times (in seconds), while s holds the respective standard deviations. Rows are associated with implementations (i.e., sorting algorithms), while columns represent setups (i.e., vector sizes). The first parameter of perfstats specifies whether to generate a scalability plot. More specifically, the value 3 orders the function to generate a semi-logarithmic plot,

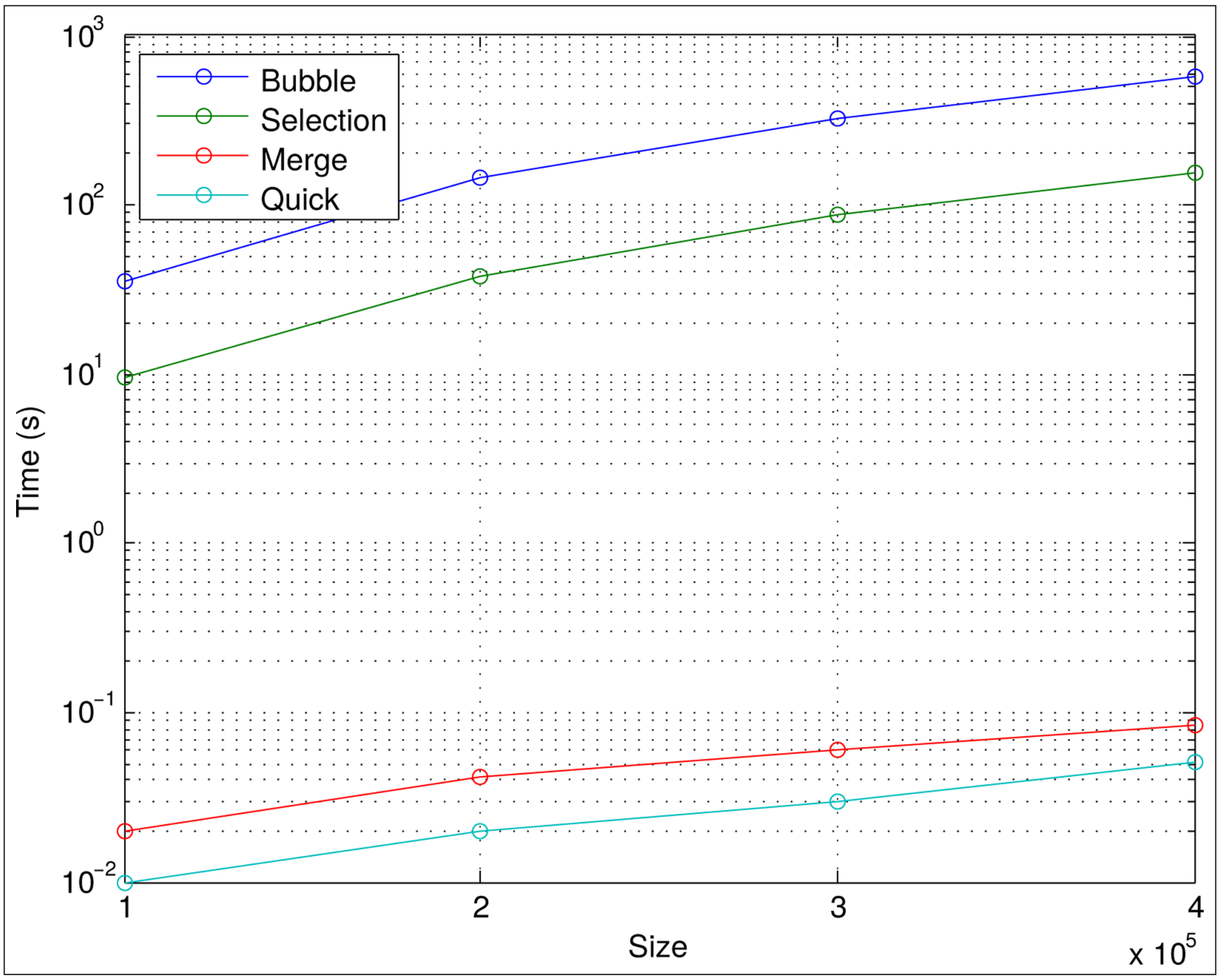

Figure 2: Scalability plot generated by the `perfstats` function.

as show in **Figure 2**. Negative values indicate that the figure should also display error bars representing the standard deviation in the measured computational sizes. No plot will be generated if zero is passed as the first argument.

Obtaining the speedup: The `speedup` function determines speedups against one or more reference implementations, across a number of setups. Its usage is similar to that of `perfstats`, requiring the identification of the implementation specs to compare:

```
[s_avg, s_max, s_min] = speedup(-2, 1, ...
    'Bubble', bs, 'Selection', ss, ...
    'Merge', ms, 'Quick', qs);
```

The first parameter concerns the optional bar plot the function is able to generate. An absolute value of `2` states that a bar plot with a logarithmic scale should be generated, as shown in **Figure 3**. Since this value is negative, error bars representing the maximum and minimum speedups are drawn on top of the average speedup bars. The second parameter defines the reference implementation(s) to which the speedups are to be determined against. Passing `1` identifies the first implementation, Bubble sort, as the reference. The `speedup` function returns cell arrays containing the average, maximum and minimum speedup matrices for each reference implementation. In this case, one reference was defined, and thus only the first item in the returned cells is available:

```
s_avg{1} =
    1.0e+04 *
    0.0001        0.0001        0.0001        0.0001
    0.0004        0.0004        0.0004        0.0004
    0.1800        0.3532        0.5420        0.6798
    0.3600        0.7241        1.0839        1.1331
```

In a similar fashion to the mean and standard deviation matrices returned by `perfstats`, rows of speedup matrices are associated with implementations (i.e., sorting algorithms), while columns represent setups (i.e., vector sizes). Note that, in this case, the first row represents the average speedup of Bubble sort against itself, and, as such, the values are all ones.

Generating tables: `PerfAndPubTools` can generate plain text or publication quality tables summarizing the performed computational benchmarks. The process is divided in two steps using the `times_table` and `times_table_f` functions, respectively. The former determines and returns a matrix containing partial or complete information to generate a table, while the latter effectively generates tables. This division is useful because `times_table_f` can accept more than one matrix returned by `times_table`, allowing the generation of more complex tables.

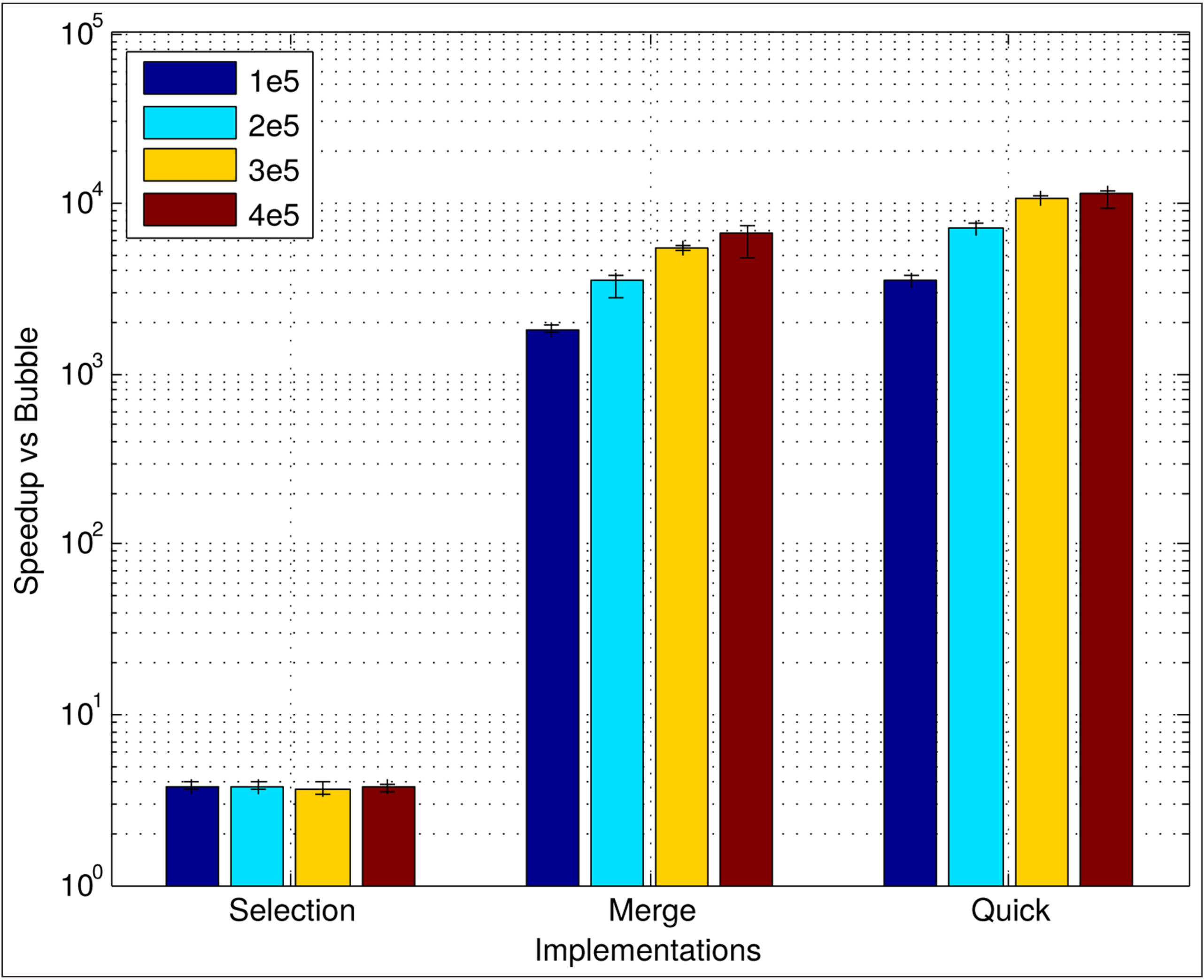

Figure 3: Speedup plot generated by the `speedup` function.

The `times_table` function, like `perfstats` and `speedup`, requires the identification of the implementation specs to compare, as shown in the following command:

```
tdata = times_table(1, ...
   'Bubble', bs,'Selection', ss, ...
   'Merge', ms, 'Quick', qs);
```

The first argument designates the references implementation or implementations, in a similar fashion to the second parameter of `speedup`. The return value, `tdata`, can be passed to `times_table_f` in order to generate a table:

```
times_table_f(0, 'vs Bubble', tdata)
```

The first argument, `0`, instructs the function to generate a plain text table, as shown in **Figure 4**. Setting this value to `1` would generate a LATEX table, as shown in **Figure 5**. Note that LATEX tables require the `siunitx`, `multirow` and `booktabs` packages.

Complete example: The complete example is available in the user manual bundled with the software. It contains the necessary steps required to reproduce these results, also showing how the return values of `perfstats` and `speedup` can be used to generate custom publication quality plots. The user manual also details an additional example concerning the performance of serial and parallel implementations of the PPHPC simulation model [3], namely different ways of contextualizing the concept of *computational size*, and the generation of more complex tables.

Quality control
The available functions are covered by unit tests in order to ensure their correct behavior. The `MOxUnit` framework [11] is required for running the unit tests. Additionally, all the examples available in the user manual (bundled with the software) have been tested in both MATLAB and Octave.

(2) Availability
Operating system
Any system capable of running MATLAB R2013a or GNU Octave 3.8.1, or higher.

Programming language
MATLAB R2013a or GNU Octave 3.8.1, or higher.

Dependencies
There are no additional dependencies for the package tools. However, unit tests depend on the `MOxUnit` unit test framework for MATLAB and GNU Octave.

```
                  ------------------------------------------------
                  |                  vs Bubble                   |
-----------------------------------------------------------------
| Imp.    | Set.    |    t(s)     |     std     |  std%   |  x Bubble  |
-----------------------------------------------------------------
| Bubble  |     1e5 |          36 |       0.887 |    2.46 |          1 |
|         |     2e5 |         145 |        2.92 |    2.02 |          1 |
|         |     3e5 |         325 |        6.19 |    1.90 |          1 |
|         |     4e5 |         578 |        6.38 |    1.10 |          1 |
-----------------------------------------------------------------
| Select  |     1e5 |        9.53 |       0.069 |    0.72 |       3.78 |
|         |     2e5 |          38 |       0.283 |    0.74 |       3.81 |
|         |     3e5 |        88.5 |         3.7 |    4.18 |       3.67 |
|         |     4e5 |         154 |        3.06 |    1.99 |       3.76 |
-----------------------------------------------------------------
|  Merge  |     1e5 |        0.02 |    3.66e-18 |    0.00 |    1.8e+03 |
|         |     2e5 |       0.041 |     0.00316 |    7.71 |   3.53e+03 |
|         |     3e5 |        0.06 |    1.46e-17 |    0.00 |   5.42e+03 |
|         |     4e5 |       0.085 |      0.0127 |   14.93 |    6.8e+03 |
-----------------------------------------------------------------
|  Quick  |     1e5 |        0.01 |    1.83e-18 |    0.00 |    3.6e+03 |
|         |     2e5 |        0.02 |    3.66e-18 |    0.00 |   7.24e+03 |
|         |     3e5 |        0.03 |    7.31e-18 |    0.00 |   1.08e+04 |
|         |     4e5 |       0.051 |     0.00316 |    6.20 |   1.13e+04 |
-----------------------------------------------------------------
```

Figure 4: Plain text table generated by `times_table_f`.

Version	Size	vs Bubble		
		$\bar{t}$(s)	s(%)	S_p^{Bubble}
Bubble	1e5	36.00	2.46	1.00
	2e5	144.82	2.02	1.00
	3e5	325.17	1.90	1.00
	4e5	577.86	1.10	1.00
Selection	1e5	9.53	0.72	3.78
	2e5	38.05	0.74	3.81
	3e5	88.51	4.18	3.67
	4e5	153.66	1.99	3.76
Merge	1e5	0.02	0.00	1800.20
	2e5	0.04	7.71	3532.22
	3e5	0.06	0.00	5419.55
	4e5	0.08	14.93	6798.35
Quick	1e5	0.01	0.00	3600.40
	2e5	0.02	0.00	7241.05
	3e5	0.03	0.00	10 839.10
	4e5	0.05	6.20	11 330.59

Figure 5: LATEX table generated by `times_table_f`.

List of contributors
The software was created by Nuno Fachada.

Software location
Archive

Name: `PerfAndPubTools`
Persistent identifier: `https://zenodo.org/record/50190`
Licence: MIT License
Publisher: Zenodo
Date published: 21/04/2016

Code repository
Name: `PerfAndPubTools`
Identifier: `https://github.com/fakenmc/perfandpubtools`
Licence: MIT License
Date published: 21/04/2016

Language
English.

(3) Reuse potential
These utilities can be used for analyzing any computational experiment. As described in 'Implementation and architecture', other benchmark data formats can be specified by implementing a custom function to replace `get_gtime` and setting its handle in the `gather_times` function. Results from `perfstats` and `speedup` functions can be used to generate other types of figures. The same is true for `times_table`, the results of which can be integrated in table layouts other than the one provided by `times_table_f`.

Competing Interests
The authors declare that they have no competing interests.

Acknowledgements
This software is enhanced by the `matlab2tikz` script and by the `siunitx`, `multirow` and `booktabs` LATEX packages.

Note

[1] e.g., files containing the output of GNU time.

References

1. **The MathWorks** 2013 Inc. Natick, Massachusetts, USA MATLAB and Statistics Toolbox Release 2013a.
2. **Eaton, J W, Bateman, D, Hauberg, S** and **Wehbring, R** 2015 GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations, CreateSpace Independent Publishing Platform, fourth edition, (March).
3. **Fachada, N, Lopes, V V, Martins, R C** and **Rosa, A C** 2016 Parallelization strategies for spatial agent-based models. *International Journal of Parallel Programming*, (January) pp. 1–33.
4. **Keppel, D, MacKenzie, D, Juul, A H** and **Pinard, F** 1990 GNU time, available: `https://www.gnu.org/software/time/`.
5. **Wright, J** 2016 siunitx: A comprehensive (SI) units package, (January) available: `https://www.ctan.org/pkg/siunitx`.
6. **van Oostrum, P, Bache, Ø** and **Leichter, J** 2010 The multi- row, bigstrut and bigdelim packages, (February) available: `https://www.ctan.org/pkg/multirow`.
7. **Fear, S** 2005 Publication quality tables in LATEX, (April) available: `https://www.ctan.org/pkg/booktabs`.
8. **Schlomer, N** 2008 matlab2tikz, available: `http://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz-matlab2tikz`.
9. **Fachada, N** 2016 Self-contained ANSI C program for benchmarking sorting algorithms, available: `https://github.com/fakenmc/sorttest_c`.
10. **Sedgewick, R** 1997 Algorithms in C, Parts 1–4: Fundamentals, Data Structures, Sorting, Searching, Addison Wesley, (September).
11. **Oosterhof, N N** 2015 MOxUnit – An xUnit framework for Matlab and GNU Octave, available: `http://www.mathworks.com/matlabcentral/fileexchange/54417-moxunit`.

How to cite this article: Fachada, N, Lopes, V V, Martins, R C and Rosa, A C 2016 PerfAndPubTools – Tools for Software Performance Analysis and Publishing of Results. *Journal of Open Research Software,* 4: e18, DOI: http://dx.doi.org/10.5334/jors.115

Submitted: 26 January 2016 **Accepted:** 26 April 2016 **Published:** 12 May 2016