PMLAB:
An Scripting Environment for Process Mining

Josep Carmona[1] and Marc Solé[2⋆]

[1] Universitat Politècnica de Catalunya, Barcelona (Spain)
[2] CA Technologies, Barcelona (Spain)

Abstract In a decade of process mining research, several algorithms have been proposed to solve particular process mining tasks. At the same pace, tools have appeared both in the academic and the commercial domains. These tools have enabled the use of process mining practices to a rather limited extent. In this paper we advocate for a change in the mentality: process mining may be an exploratory discipline, and PMLAB – a Python-based scripting environment supporting this – is proposed. This demo presents the main features of the PMLAB environment

1 Introduction

In *scientific computing*, one is not only building computational models and complex algorithms that enable quantitative analysis, but also continuously exposing these models and algorithms to real data in order to have a better understanding of the reality being studied. This *exploratory* view of the field (algorithms help but nobody in the field assumes that are sufficient to solve their particular problems) made environments like MATLAB® or Mathematica® to be tremendously successful in helping the progress of research. We advocate for having a similar environment in the novel discipline of *process mining*.

In a nutshell, we believe process mining should be programmed and not only used. There are several tools available to use process mining algorithms, being ProM [3] the state-of-the-art tool. ProM is a great academic effort that incorporates around three hundreds plugins programmed from very different universities. It allows normal users, i.e., those ones not familiar with process mining, to use a graphical user-friendly front-end to process mining algorithms. This strength (process mining algorithms accessible for the masses) has become, in our opinion, a weakness: the expert user is restricted to work with strict GUIs, and it takes a considerable effort when the particular task to achieve is not fully satisfying the requirements of the plugins. As a programmer, a deep knowledge of the internals of ProM is required in order to create a new plugin, even if it represents a slight modification of the ones available.

PMLAB is an interactive programming environment for (exploratory) process mining computing and/or research on top of a process-oriented language. In this

language, *logs*, *models* and many other high-level objects/tasks are first-class citizens, meaning that one can compute (interactively or not) on the basis of these elements. Importantly, there can be different granularities on the view of these high-level elements, e.g., a log can be simply passed to a discovery algorithm (*coarse-level view*), or analyzed to derive the most frequent cases (*introspective view*). The following is a list of PMLAB features:

- *Interactive shell*: as happens in Mathematica, a shell where every object used/computed is available is provided, and process mining algorithms may be applied to these objects to create new ones. The typical session may start by importing the libraries to be used, and to continuously enrich the environment by computing new objects from the existing ones.
- *Process mining elements as first-class citizens of the language*: importantly, the environment offers a solid and consistent library for some of the main tasks required in process mining, e.g., importing a log in XES format. Once a log is imported into a variable, algorithms can be applied on the variable to produce new elements (e.g., a discovery algorithm to derive a BPMN model).
- *Programmer friendly*: the environment not only provides the necessary help for using the elements, but more importantly describes them in a way a programmer can incorporate these objects onto her/his programs.
- *Extendable*: new functionalities can be added by means of new library modules.
- *Irredundant*: to have thirty algorithms to perform the same task maybe is not the ideal situation for using that functionality. As a policy, we believe the core environment should limit the amount of redundancy in order to simplify the usage.
- *Simple Programming*: the syntax and semantics of the language should be easy, in order to allow for easy programming. One example of this is types in programming languages: although useful for programming and compilation, the learning curve required to master a statically-typed language is significantly higher than the one for a dynamically-typed language. This makes dynamically-typed languages as Python a good candidate.
- *OS exposed*: there is a good marriage between the operating system elements (files, directories, databases, etc ...) and the elements of the environment. This will easy the management and manipulation of the data within the environment.
- *With support to distributed/parallel computing*: it is fairly easy to distribute or parallelize the computations to take advantage of the computing resources available.

2 PMLAB Tool Description

2.1 Architechture

Figure 1 describes informally the three computing resources available in the environment:

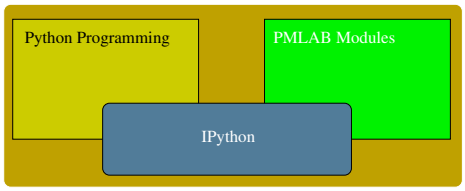

Figure 1. The `PMLAB` environment.

- `IPython` [1]: this environment offers the shell to perform the computations needed to carry out process mining tasks. Many of the functionalities required in the previous section exist in IPython. It is an open-source environment (license BSD).
- `PMLAB` modules: a set of process mining modules will form the basis for process mining computing. These modules will contain algorithms for the three process mining dimensions: discovery, conformance and enhancement. In the current version of PMLAB, only discovery and enhancement algorithms are included.
- Python Programming: finally, for any task not considered in the `PMLAB` modules, one can always use a python program on top of IPython. The results and intermediate computations (e.g., program's variable assignments) may be, if wanted, incorporated into the IPython shell after running the program.

In the next subsection we provide an example illustrating the user perspective of `PMLAB`.

2.2 Maturity

The tool is currently under development, and therefore it may evolve towards a more stable version in the near future. The current version can be seen as a prototype of the ideas underlying scripting process mining algorithms. Moreover, the tool has only been tested with small or medium sized examples. In spite of this, there are few universities and companies using the tool in its current state.

In terms of features, the tools provides support for the following objects: logs, Petri nets, transition systems, Causal nets, BPMN models. There exist transformations between some of these elements (e.g. from Causal nets to BPMN, or from logs to transition systems), and discovery techniques for Petri nets, Causal nets, and BPMN models. All elements can be graphically visualized and some of them simulated. High-level algorithms like log clustering, filtering, projecting and event encoding are also available. Since PMLAB supports some of the standard process mining formats, it can be used to interact with other tools.

2.3 Getting the Tool

In the following web page: `http://www.lsi.upc.edu/~jcarmona/PMLAB/`

one can find all the required information: a tutorial including installation instructions and the distribution. Currently the tool is distributed in two forms:

- Virtualized: We have created a VirtualBox virtual machine in `Lubuntu` which can be easily downloaded and installed in few steps.
- Sources: We provide the python library together with the installation script. It is expected to be installed in a `Linux` distribution, since some binaries that are also provided are compiled for this platform.

3 Example

A session that uses the already available functionalities in the environment is now described. We simply begin by starting the IPython environment in a directory that contains a XES file named `exercise5.xes`, which belongs to the example files distributed with ProM. The log contains sequences representing the typical actions in the process of reviewing papers for scientific publication.

Our first action is loading the module that handles the logs, and reading the file.

```
>>> import pmlab.log
>>> log = pmlab.log.log_from_file('exercise5.xes')
>>> log.statistics()
Alphabet size: 14
Number of cases: 100
Number of unique cases: 96
Length of shortest case: 11
Length of largest case: 50
Average case length: 23.0
```

Imagine that we want to communicate the model to a company whose members are only familiar with the BPMN notation. The `PMLAB` package has a module that allows discovering BPMN diagrams from C-nets [2], so first of all we will discover a C-net and then transform it into a BPMN diagram. To discover a C-net we must load the C-net module and condition the log (C-net have some particular conditions that have to be fulfilled by the logs). The corresponding instructions are shown below:

```
>>> import pmlab.cnet
>>> clog = pmlab.cnet.condition_log_for_cnet(log)
>>> cn, bfreq = pmlab.cnet.cnet_from_log(clog)
>>> cn.save('exercise5.cn')
>>> pmlab.cnet.save_frequencies(bfreq, 'exercise5.bfreq')
```

Additionally, after discovering the C-net we have saved the net and the binding frequencies discovered in two files just in case we need them in another occasion.

Finally we will transform the C-net into a BPMN diagram. To do so we will first load the appropriate module and call the transformation function. Then we will add the frequency information to the diagram (so that most frequent paths appear thicker than infrequent ones), saving the diagram in a graphviz DOT file.

```
>>> import pmlab.bpmn
>>> bp = pmlab.bpmn.bpmn_from_cnet(cn)
>>> bp.add_frequency_info(clog, bfreq)
>>> bp.print_dot('exercise5freq.bpmn.dot')
```

Then the DOT file can be used to generate a graphic file using the Graphviz suite, that can be called directly from IPython using the following command:

```
>>> !dot -Tps exercise5freq.bpmn.dot > exercise5freq.bpmn.ps
```

That produces the BPMN diagram of Figure 2 (bottom).

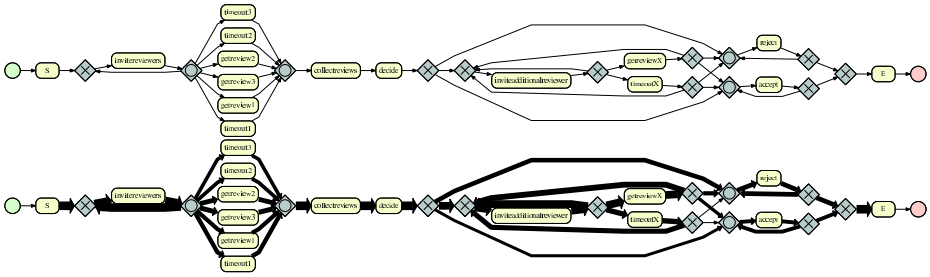

Figure 2. BPMN diagrams of the example produced by the tool: without frequencies (top), with frequencies (bottom).

Up to this point we have shown a classical use of the environment as a simple user. However, for this kind of tasks, a more user-friendly GUI would be nicer and would also save typing. What are the advantages of the environment ? Let us illustrate some of them. Assume that you want to repeat the previous processing with 1,000 different files. No problem. IPython offers a save command in which you can indicate which instruction numbers you want to save to a file. Using that command we can save all the previous typed instructions to a text file named, for instance, `discoverBPMN.py`. Now in this script we can change the literal `'exercise5.xes'` for a variable inside a loop that takes as value the name of each one of the 1,000 files. This script can be executed inside IPython with a simple `run discoverBPMN.py`

References

1. F. Pérez and B. E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.
2. W. van der Aalst, A. Adriansyah, and B. van Dongen. Causal nets: a modeling language tailored towards process discovery. In *CONCUR*, pages 28–42, 2011.
3. W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, and T. Weijters. Prom: The process mining toolkit. In A. K. A. de Medeiros and B. Weber, editors, *BPM (Demos)*, volume 489 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.