Literate Programming and Beyond

Carlos Linares López*

May 30, 2017

Abstract

Literate Programming is a rather old paradigm originally introduced by Donald Knuth in the 80s. All in all, the idea consists of providing a better support to the documentation of source code. Admittedly, this paradigm requires software tools to support it specifically, yet the approach is simple and very efficient. Furthermore, this paradigm has motivated other techinques.

Introduction

This document and all the files distributed along result from an Innovation Project awarded to the subject *Heuristics and Optimization*. Essentially, this project proposes the usage of new (and efficient) documentation techniques for providing additional support for different software.

One of the main drivers in selecting these documentation techniques is to be *language-independent*, i.e., for it to be able to provide support to a plethora of programming languages. The reason is that programming languages are decoupled from the algorithm —certainly, one could provide a pseudocode that can be read in a specific language, but ideally the algorithm should never be tied to specific implementation details. And truth to be told, the subject you have to study this year is all about algorithms. You are now about to focus on different ideas for their implementation.

One example is `iPython` which was originally conceived to allow the reader to both read the document, and also to try different implementations in arenas where they could be executed (either interpreted/evaluated

*carlos.linares@uc3m.es

or compiled and then executed). The idea is so powerful that other programming languages provide similar resources, such as Go [1]. While the results are truly impressive [2], they were restricted only to the Python programming language. Therefore, the idea behind `iPython` was pushed further by showing compatibility with many different programming languages. The result is known as `iJupyter` [3] and it is *de facto* language-independent.

Another example is *Literate Programming* that was advocated by Donald Knuth as a better way to programming. Indeed, Donald Knuth wrote: "*[...] it seems to me that at last I'm able to write programs as they should be written.*". In a nutshell, the idea consists of creating files that contain both an implementation and also its documentation. It is then possible to perform two different operations:

- *Tangling*: consists of extracting the source code.
- *Weaving*: produces the documentation in a readable format, such as pdf.

There are a good number of software that provide support to this idea. One of the most remarkable editors, Emacs [4], does as well. As a matter of fact, Emacs goes well beyond the principles of Literate Programming and provides also support to a far more advanced concept, *Reproducible Research* though the `org-babel` mode —which unfortunatelly falls out of the scope of this document.

The following two sections describe the material that has been produced in the context of the current *Innovation Project*:

- `iJupyter` has been used for documenting `PySimplex`, a software tool for solving Linear Programming tasks;
- *Literate Programming* in Emacs is exemplified with the documentation of various informed and uninformed search algorithms.

[1] For a better understanding, the interested reader is refered to the Golang Tour (`https://tour.golang.org/`) where it is possible to try different programs in Go along the documentation provided for each case.

[2] See, for example, `https://plot.ly/ipython-notebooks/`

[3] See `http://jupyter.org/`

[4] See `https://www.gnu.org/software/emacs/`

Linear Programming

This project provides access to `PySimplex`, a Python implementation of the SIMPLEX method studied in the syllabus, by Carlos Clavero Muñoz, as part of his TFG. In case the specified Linear Programming task has only two decision variables, then the *graphical resolution* method can be enabled as well.

This implementation is not intended to be efficient in any way or matter. Instead, it implements SIMPLEX exactly as explained in the lectures. Additionally, it produces the same output. This approach provides two immediate benefits:

1. It should be very easy for any student to use the software: from specifying linear programming tasks to perfectly understand the output produced by it.

2. It can then be used for making additional exercises. Indeed, it is now feasible to solve an infinite number of cases, as many as the student is interested in solving.

The software is available in `github` [5]. Remarkably, the project uses `iJupyter` for documenting its usage [6]. Besides all these resources, there is also a video available (in Spanish) which provides a quick tour to `PySimplex` [7].

Search algorithms

Along with this document a *tarball* is distributed also with various implementations of both informed and uninformed search algorithms. All the algorithms are implemented and documented in an `.org` file which can be used to either *weaving* the documentation or to *tangling* the source code.

Of course, all students are strongly encouraged to use the `.org` files as it discusses the documentation in detail along with the source code itself. These files are human readable and should not pose any difficulty to be well understood. As a matter of fact, reading these files can serve as a good introduction to the org mode in Emacs.

[5] See `https://carlosclavero.github.io/PySimplex/`

[6] For example, check out any of the Jupyter notebooks in the documentation folder in Github `https://github.com/carlosclavero/PySimplex/tree/master/Documentation`

[7] Distributed along this entry in Aula Global

When uncompressed, the tarball will show two directories (among others):

`uninformed/` It contains a directory for each *brute-force* search algorithm implemented and documented with Literate Programming

`informed/` It shows up a subdirectory for each *heuristic* search algorithm

In all cases, each algorithm is implemented under a different subdirectory which always follow the same structure:

`org/` Contains the org file used to generate both the documentation and the code

`doc/` Contains the documentation properly formatted as a pdf file, as a result of *weaving* the org file.

`code/` Contains all the code that results from *tangling* the org file.

The different implementations provided in this project have been developed, intentionally, in different programming languages: Go, C and Python. There is no intention whatsoever in providing efficient implementations but to discuss the implementation of each algorithm in a systematic and pedagogical way. These presentations should serve, indeed, for accomplishing the lab assignment on search algorithms without major difficulties.

About this document

As an additional demonstration of the power of Literate Programming, this document itself has been generated using Emacs under the so-called org mode. Even if it contains no source code, a file (`intro.org`) has been generated with the same contents shown in the pdf file you are reading right now. As you start reading the documentation in the tarball attached you will notice that the format of this document is different than that. This one is indeed much simpler: it contains no bibliographical references, it is one-column, contains no additional fonts, etc. However, it serves to most purposes.

For the sake of completeness the file `intro.org` (under the directory `intro/`) is included as well in the attached tarball, and can be used for further reference.