

PyAlg: An Algorithm Learning Platform ^{*}

Radu Drăgușin, Paula Petcu

Department of Computer Science
University of Copenhagen, Denmark
E-mail: petcu@diku.dk

Abstract

The course on algorithms and data structures is a fundamental course for those studying computer science. Thus, a huge amount of teaching material covering this topic has been developed over the years: from books on algorithms and data structures and articles on algorithm learning, to algorithm libraries and algorithm-visualization tools. However, our study of previous work shows the lack of a unitary solution that covers essential aspects of algorithm learning: algorithm libraries, visualization, and benchmarking. The aim of this project was to develop a unified interface covering those aspects. The developed tool, named PyAlg, is a learning platform that can be used in studying, teaching, and analysing algorithms. It was originally designed for the students following the undergraduate course on algorithms and data structures in the Department of Computer Science at the University of Copenhagen, where it was used in one of the programming assignments. The target audience for the latest version of the developed product is wider, though the objective remains the same: helping students in the process of understanding, learning, and analysing algorithms.

Keywords: Algorithms and data structures, Algorithm benchmarking, Algorithm libraries, Python, Programming language

1 Introduction

1.1 Motivation

The undergraduate course *Algorithms and Data Structures* in the Department of Computer Science at the University of Copenhagen is a nine-weeks course given for first year students studying computer science or mathematics. In the previous iterations of the course it was observed that the students had two types of difficulties in understanding the course material. The students from the computer-science department had problems with understanding the proofs of the correctness and performance of algorithms, and often skip formulas and text, and just read the pseudo-code. On the other hand, the students with a mathematical background had problems with seeing the connection between the pseudo-code and real programs run on computers.

^{*} © 2010 International Conference on Virtual Learning. Proceedings of ICVL (Print ISSN 1844-8933), Bucharest, Romania. This is the authors' version of the work. The original publication is available at <http://c3.icvl.eu>.

Starting from 2010, the purpose of the course was to cover the whole algorithm-engineering cycle (Sanders, 2009): design, analysis, implementation, and experimentation.

However, many of the students enrolled have never programmed before. Based on the instructors' previous experience, the Python programming language was chosen for the programming assignment for this course. The literature on teaching algorithms supports their choice (Stajano, 2000; Chou, 2002; Miller and Ranum, 2005). Python is an easy-to-learn programming language, with clear syntax and an extensive library. Moreover, using Python can prove to be much more productive compared to other programming languages.

1.2 Project Overview

The aim of our project was to facilitate the learning of algorithms by creating a playground for the first-year students enrolled in the course on algorithms and data structures. The final output was an application, named PyAlg, which can be used as a learning platform supporting the lectures or self-study. It was originally designed for the students following the undergraduate course in the Department of Computer Science at the University of Copenhagen, but could be used in other institutions as well.

The lectures and exercise classes provided by the instructors usually cover only the design and analysis parts of the algorithm-engineering process (DFG, 2007). The aim of this project was to provide a learning platform, which supports the implementation and experimentation parts of the process.

Although the current main features of the application are developing, organizing, and benchmarking algorithms, the underlying target of the application is to give students a good start in understanding, analysing and extracting relevant information pertaining to algorithms. The application could also be used as a tool supporting algorithm research.

2 State of the Art

In the Department of Computer Science at the University of Copenhagen, the widely-used *Introduction to Algorithms* textbook (Cormen et al, 2009) has been used since the mid 90s. The book is used both as reading material and as a source of assignments for the students. However, for the hands-on programming exercises that should cover the implementation and experimentation part of the algorithm-engineering process, new material was needed.

The CPH STL (www.cphstl.dk), developed and maintained by the Performance Engineering Laboratory from the same department, is a library of generic algorithms and data structures implemented in C++; this library provides alternative/enhanced versions of the individual C++ standard library components using standard algorithmic and performance-engineering techniques (Katajainen and Yde, 2000). LEDA (Library of Efficient Data types and Algorithms) also comprises of an extensive collection of data structures implemented in C++ (Mehlhorn and Näher, 1999).

The AlgoViz Wiki project (Shaffer, 2009) provides information to support users and developers of algorithm visualizations, and includes a catalogue of publicly available algorithm visualizations. However, they only provide links to different

sources of algorithm visualizations, rather than actually hosting interconnected visualizations. On the other hand, the algorithm animator, developed by Massimo Di Pierro in conjunction with the course *Design and Analysis of Algorithms* at DePaul University, seems to capture the visualization of most of the algorithms studied in a basic course on algorithms and data structures. It is a “Python application that implements and animates interactively those algorithms that are normally covered in an undergraduate course on the topic” (Di Pierro, 2008). For most algorithms, the source code actually follows the pseudo-code from the textbook *Introduction to Algorithms*.

Another type of application that contributes to the experimental part of algorithm-engineering is a benchmarking tool. For example, Benchrun (Johansson, 2010) is a Python script for defining and running performance benchmarks for different versions of some code for different values of an input parameter.

Although several sources of teaching and study material are available, we did not find a complete application that includes all three components our project focuses on: algorithm library, algorithm visualization, and algorithm benchmarking.

3 Software Solution

3.1 Overview

The features offered by our application are algorithm benchmarking, execution-time and line-count comparisons, control-flow visualization, and algorithm-library management. The name of the application, PyAlg, comes from the concept of providing an *algorithms playground* in *Python*.

In what follows, we will provide a general overview of the application. Detailed information on usage can be found on the project’s website (Drăgușin and Petcu, 2010).

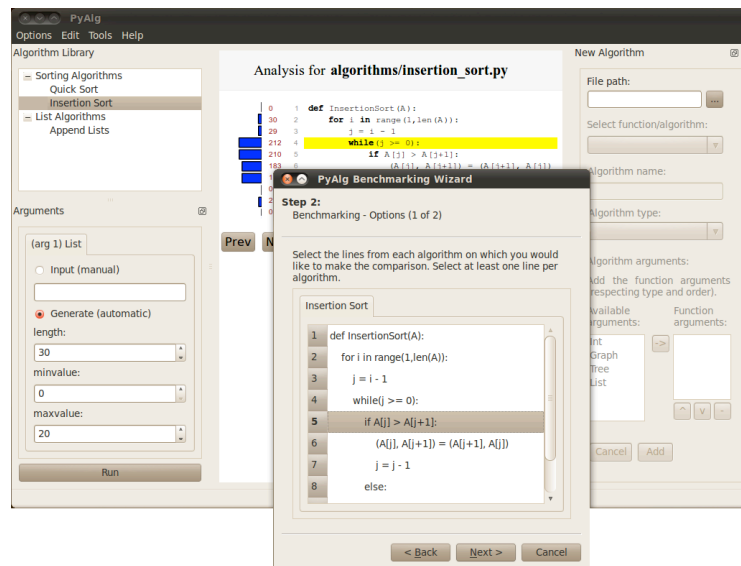


Figure 1: PyAlg interface on Ubuntu 10.04

The main functionalities of the application can be observed in Figure 1, illustrating the interface of the application. As previously mentioned, our project focuses on two

specific key aspects: a library of algorithms and a benchmarking tool. Another team developed an algorithm visualization tool (Juncher and Mathiasen, 2010), which is not integrated in the current version of the project.

3.1.1 Algorithm Library

The leftmost area of the interface includes a list of algorithm names that have been previously introduced. In our examples, the application has initially two sorting algorithms, but the user can easily add more algorithms to the library and group them under custom sections.

The central area of the main window renders a dynamically-created HTML file with embedded JavaScript. The user can run an implementation of the algorithm from the library on randomly generated values or on user given values. The selection of the values for input arguments is done through the input area from the bottom-left corner, as depicted in Figure 1. An HTML file corresponding to the selection is generated each time the user runs the program through this interface.

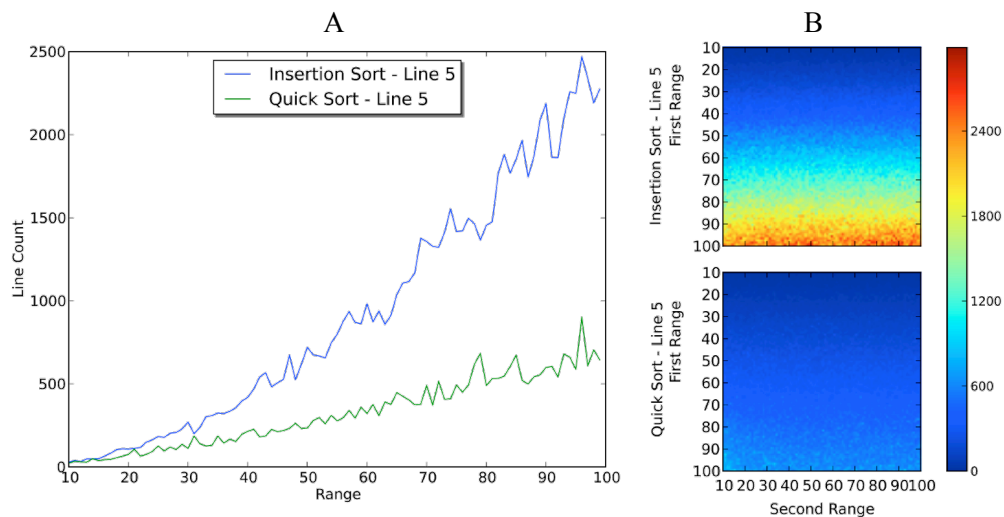


Figure 2: Benchmarking using PyAlg

(A) The image is the result of benchmarking Insertion Sort (blue) and Quick Sort (green) on 90 lists of increasing length (ranging from 10 to 100 elements – x-axis). The y-axis represents the number of comparison performed to sort each list. (B) The image adds one more range element – the maximum size of the elements of the list, on the x-axis – besides the length of the list on the y-axis. The number of comparisons is represented through colour. A student could easily observe that once the length of a list increases, the number of comparisons in Insertion Sort grows considerably faster than in Quick Sort (A and B) and that the maximum size of the elements of the list does not influence the number of comparisons (B).

The generated HTML file provides a simple analysis of the used resources and control flow and can be saved through the interface for further use. The file contains

the Python source code with syntax highlighting, the associated line numbers and line counts (both numerically and graphically), and the highlighted control flow of the algorithm (which can be controlled through buttons). However, for more complex analysis and comparisons, the user can use the benchmarking tool.

3.1.2 Benchmarking Tool

The benchmarking tool allows the user to compare and visualize the time needed for an operation to be executed by different algorithms. Moreover, the computational cost of these operations can be observed over a range of input parameters. The students can also use the benchmarking tool to establish which is the most computationally expensive operation in an algorithm.

The user must follow three steps: select one or more algorithms, select one or more lines from each algorithm, and provide values for the arguments of the algorithms. An example output is shown in the Figure 2.

3.2 Teaching and Learning Using PyAlg

The software has the goal of facilitating teaching, learning, and experimenting with algorithms in class and at home.

Through the use of PyAlg, the teacher can conceive, design and then generate support materials for use in class. The teacher can supplement the lecture with graphical representations produced by the application. This can be of use in a number of occasions, for example, when comparing algorithms, or in establishing their computational complexity.

One of the first features we designed was to allow users to add new algorithms through the interface. We wanted to make this feature as flexible as we could so that the user can add any Python code. The source code does not need to contain an algorithm usually taught at algorithmics courses; it can be any Python code. Moreover, in order to benchmark an algorithm or analyse it, no further modifications to the source code are needed. Thus, the students can use the application to experiment on their own code and test the performance of their algorithms.

We chose to produce the output for the algorithm analysis in the form of an HTML file. This file can be viewed in the interface, as well as be saved and used outside the scope of the application. The corresponding image for line counts is saved in the scalable vector graphics (SVG) format. The output images of the benchmarking tool are also saved in the SVG format. Also, when saving these images, the raw data on which the programs run during benchmarking is also saved in a comma-separated values (CSV) file, facilitating further data processing. Moreover, this feature allows students and teachers to easily integrate benchmarks into their presentations, lectures, or reports.

We also chose to facilitate the interaction with all of the provided functionalities through an easy-to-use graphical interface rather than through command-line instructions. This decision was based on the consideration that the target audience is formed of computer-science students as well as students from other fields.

We felt that it was important to allow students to work on the operating system they are familiar with, so our application works on all major operating systems (Linux, Windows, Mac OS X and FreeBSD).

The application is almost entirely written in Python, the exception being the JavaScript file that is responsible for highlighting the control flow of an algorithm in the produced HTML file. The project is open-source, being licensed under GNU GPL v.2. Thus, it can be used or further extended by other institutions or individuals that have an interest in it.

3.3 PyAlg in Class

The instructors saw the project assignment as a link in the algorithm-engineering cycle, covering the implementation and experimenting phases, as illustrated in Figure 3.

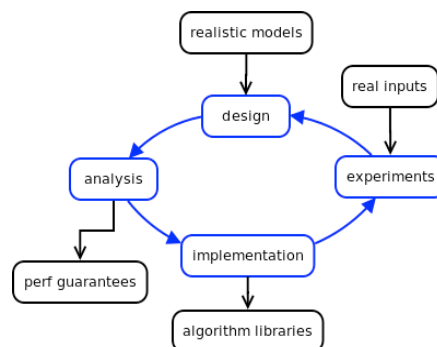


Figure 3: The cycle of algorithm-engineering: design, analysis, implementation, and experimental evaluation of algorithms. While the lectures covered design and analysis, the project covers implementation and experimentation. The illustration is based on (DFG, 2007).

The students had two weeks to solve the project, which consisted of the tasks of implementing a specified data structure and algorithm, writing a test suite for verifying the functionalities of the written code, benchmarking the performance of their code and some competing code using PyAlg, and finally visualizing how the data structure or algorithm behaves.

Prior to the assignment, the students were offered the opportunity to attend a *Quick Start to Python* two-hours class. The overall content and purpose for these classes was to introduce the Python programming language and get the students to start programming.

4 Conclusions and Future Work

4.1 Conclusions

The main objective of our project was to help students in the process of understanding, learning, and analysing algorithms. Together with the visualization toolbox VisPy (Juncher and Mathiasen, 2010), we developed a complete product that includes all the three components our project initially intended to focus on: an algorithm library, an algorithm visualization tool, and an algorithm benchmarking tool.

The target audience of the developed product is wide and the existing framework should be able to be successfully operated as it is in its current version by any individual with a little experience in Python programming.

The development of PyAlg was part of an integrated effort to help students in learning algorithms and to motivate them for this goal. Our own experience as graduates of the West University of Timisoara is that the desire to make algorithm courses more appealing to first-year students is not limited to the University of Copenhagen. Indeed, there are numerous reports indicating that other educational institutions seek to improve their algorithm teaching practices too (McCracken et al, 2001; Denning and McGettrick, 2005).

Our study of previous work shows the lack of a unitary solution that covers all the aspects of algorithm engineering. With this project we showed that it is possible to develop a unified interface that covers all the above-mentioned aspects.

4.2 Future Work

Although the project can be successfully used in its current form, there is always room for improvement. On the webpage of the project, we collect information on the usage of the tool and any reported problems. However, besides any issues that might be discovered in the close future, we already have some ideas for improving and expanding our application.

The first major enhancement would be to integrate a visualization toolbox inside the application as opposed to a separate package. This would allow students to analyse and visualize their algorithms in a unitary interface. Another enhancement would be the development of a web service for algorithm benchmarking that would use the core of PyAlg as a back-end. An on-line service could prove to be practical when benchmarking algorithms with large input data. This idea could be easily implemented as a consequence of PyAlg's use of web standards for output.

In the future, we plan to advertise PyAlg in the academic as well as open-source circles as a useful tool to teach, learn, and analyse algorithms.

5 Acknowledgements

We are thankful to Jyrki Katajainen for his assistance in the project and his help in revising the paper. We would also like to thank the students from the University of Copenhagen for providing us with their valuable input.

6 References

- Chou, H. P. (2002): Algorithm education in Python. In *Proceedings of the 10th International Python Conference*, pp.177-185.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009): *Introduction to Algorithms*, 3rd Edition. The MIT Press.
- Denning, P. J. and McGettrick, A. (2005): Recentering computer science. In *Commun. ACM* 48(11), 15–19.
- DFG - Deutsche Forschungsgemeinschaft (2007): DFG Priority Programme 1307: Algorithm Engineering. Retrieved June 5, 2010 from <http://www.algorithm-engineering.de/?language=en>
- Di Pierro, M. (2008): Algorithm animator source code and download page. Retrieved June 5, 2010 from <https://launchpad.net/algorithms-animator>
- Drăgușin, R. and Petcu, P. (2010): PyAlg project hosted on Google Code. Downloads List. Retrieved June 6, 2010, from <http://code.google.com/p/pyalg/downloads/list>

- Johansson, F. (2010): Benchrun: Python benchmarking utility. Retrieved June 5, 2010, from <http://code.google.com/p/benchrun/>
- Juncher, K. L. and Mathiasen, B. S. (2010): VisPy project hosted on Google Code. Vispy: tool for visualizing data structures. Retrieved June 6, 2010 from <http://code.google.com/p/vispy/>
- Katajainen, J. and Yde, L. (2000): Project proposal: The Copenhagen STL. Technical Report: CPH STL Report 2000-1, Department of Computer Science, University of Copenhagen. <http://www.cphstl.dk/WWW/mission.html>
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001): A multi-national, multi-institutional study of assessment of programming skills of first-year computer science students. *SIGCSE Bull*, 33(4), 125–180.
- Mehlhorn, K., Näher, S. (1999): LEDA: a platform for combinatorial and geometric computing, Cambridge University Press.
- Miller, B. N. and Ranum D. L. (2005): Teaching an introductory computer science sequence with Python. In *Proceedings of the Midwest Instructional Computing Symposium*.
- Sanders, P. (2009): Algorithm engineering – an attempt at a definition. In *Lecture Notes in Computer Science*, Volume 5760, Springer.
- Shaffer, C. (2009): Data structure and algorithm visualization for computer science education, <http://algoviz.cs.vt.edu/AlgovizWiki/>
- Stajano, F. (2000): Python in education: Raising a generation of native speakers. In *Proceedings of the 8th International Python Conference*, pp. 24-27.