Moopec: A Tool for Creating Programming Problems

Rui C. Mendes
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Abstract

This paper presents a tool called Mooshak ProblEm Creator (Moopec) for facilitating the creation of programming exercises for a web-based multi-site programming contest system called Mooshak [7]. Users only need to create a text file for specifying all the information concerning problems including their description, tests and user feedback. This tool provides ways of automating most tasks involved in creating problems in Mooshak and, consequently, increases teachers' productivity. Moopec allows instructors to quickly create problem sets by simply editing a text file. Moopec is implemented in Python and is available at `https://github.com/rcm/mooshak_problem_creator`.

2012 ACM Subject Classification Applied computing → Computer-managed instruction

Keywords and phrases Automatic Program Assessment, Batch Generation, Testing

Digital Object Identifier 10.4230/OASIcs.ICPEC.2021.9

Funding *Rui C. Mendes*: This research has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

Acknowledgements I want to thank Pedro Ribeiro for gently supplying the script used for providing the feedback shown in Fig. 1.

1 Introduction

Teaching students how to program is difficult [6]. There are several tasks of equal importance:
(1) Supplying enough exercises for practice;
(2) Providing feedback;
(3) Summative assessment.
With the advent of the pandemic, there is an added difficulty: instructors and students do not share the same space and thus it is harder to help them in the classroom. Thus, the importance of using software tools that are capable of performing these tasks increased. Even though there are many problems available online, instructors still need to create new problems because most of the classical exercises have their solutions publicly available online. While this is extremely useful for learning how to program, it creates many difficulties for instructors when they are interested in summative assessment. Furthermore, when students submit their solutions to an online tool, instructors are able to address other concerns like code organization, cleanness, readability, efficiency or documentation.

When teaching programming, there are several kinds of exercises that can be given to students [1] involving all the categories of the Bloom taxonomy [13]. Most of these tasks detail several ways of evaluating students in written tests. However, in this work we are interested in programming exercises that may be given to students in practical classes and be evaluated by a programming assessment tool. The rationale behind this is to enable students to work at their own leisure, in class or at home. In this way, students may try the exercises, get automatic feedback and still be able to consult instructors either online or during class time. Programming exercises may fit into several categories [11, 9]:

code from scratch where students implement a solution to a problem given a description and a test suite that assesses their performance;

implement an algorithm students are given an algorithm and are asked to implement it;
code skeleton students start with a skeleton that they have to fill, they may have to implement a few functions or classes;
code baseline the instructor supplies some code and the students have to extend it for a more general case or use it for solving a more complicated task;
jumbled code students are given all the lines of code but they are not in the correct order;
find the bug students are given buggy code and must find the bug, this is more useful when using quizzes and forces students to look at a solution, read it and try to understand why it doesn't work;
buggy code this is a generalization of the previous approach and allows students to improve their skills of code reading and understanding and can also help them become more proficient with a debugger, understand compilation errors, logic errors and problems handling edge cases ;
compiling errors this is also useful in quizzes as students are given a code snippet and the compiler error and must understand what the problem is
keyword use students are forced to use a given function or code and solve a task according to a strict specification (e.g., use `foldl` to find the maximum of a list of numbers).

There are many tools available for automatic assessment of programming exercises. Some are freely available online like codeboard [2] that provides a flexible way of creating problems and allows to implement many of the types of exercises given above. Furthermore, students can use an IDE for programming and can test their code locally or submit it. The author has used it in a course and the student's reception was good. In languages that have unit testing libraries, it provides ways of using unit tests to evaluate submissions. However, it is necessary to implement a fair portion of code for evaluating solutions in some programming languages (e.g., C). The instructor has to use a web based client and copy and paste code into the appropriate windows and create the necessary files one by one by using the WEB interface. While it is somewhat intuitive to use, its main drawback is that it does not have, as far as the author knows, a way of creating and importing problems in a batch fashion.

2 Mooshak for automatic assessment

Because of the effort necessary for creating large numbers of problems efficiently in the tools presented above, the author prefers using Mooshak [7] for automatic assessment. This is mainly due to the fact that instructors install it on a machine they control and can use the file system for creating contests, copying them, copying problems between contests and setting them up by editing text files. Mooshak has its own internal format to describe problems called Mooshak Exchange Format (MEF). MEF includes a XML manifest file referring several types of resources such as problem statements (e.g. PDF, HTML), image files, input/output test files, correctors (static and dynamic) and solution programs. The manifest also allows the inclusion of feedback and points associated to each test [10, 8]. Mooshak has been used on ICPC and IOI contests for many years as well as for education in several universities. The administrator can create several kinds of contests like:
IOI the format used in the International Olympiads in Informatics [5], where students are given points for each test they pass
ICPC the format used in the International Collegiate Programming Contest [4]
short the format used in code golf

Since its inception, Mooshak has evolved and has provided some improvements that facilitate its use in educational activities. It is possible to specify how much CPU time is necessary to solve the problem, how many resources are used and assign a point value for

each test or show a feedback message if the user fails a given test or even to show the test to the user. Each test has:

- an input, which is a text file that is fed to the program's *standard input*;
- an output, which is a text file that is compared using `diff` to the program's *standard output*;
- arguments, that are passed to the program;
- a point value, that is awarded if the submission gives the intended output for this test;
- a feedback message and
- a text file containing a context that can be used by Mooshak's dynamic evaluator for the given problem.

On the surface, it seems that Mooshak may only be used for exercise types like *code from scratch* but it may actually be used for more categories with a little imagination. Mooshak allows the administrator to configure more programming languages and to supply static or dynamic correctors for problems.

When creating a programming language, the administrator provides among others:

(1) What is the extension of the file submitted by the user

(2) How to *compile* the program

(3) How to run the program

When compiling the program, the administrator can supply the compiler with information about the team, the program environment, the extension and a solution to the problem. The static corrector receives the source code, a solution and a possible program environment and can classify the submission by simply looking at the source code. The dynamic corrector is invoked by Mooshak after it has run the program and receives information about the test like the input, expected and obtained output, the standard error contents, Mooshak's classification and the context given.

Thus, it is possible to implement some of the categories supplied above:

code from scratch this one is self-evident

implement an algorithm this category may be possible if the algorithm is the only one capable of solving the problem within a given order of complexity by careful creation of the tests and specification of the timeout in CPU seconds or the memory bound needed to solve each test

code skeleton the *compiler* may incorporate the students' solution in a larger program and thus guarantee that they actually used the skeleton given; alternatively, a static corrector could be used to check for the existence of the skeleton

code baseline same as the previous category

jumbled code the static corrector may be used for ensuring that the students didn't modify the program but simply supplied the lines in the correct order

find the bug the static corrector may be used for checking if the students are submitting the code with the bugs corrected

buggy code this is similar to the previous category

compiling errors this one may also be handled like the previous alternatives

keyword use a static corrector may help, or a specific language that logs the use of the function or keyword in the manner expected

3 Creating problems in Mooshak using the WEB interface

The difficulty involved in creating a new programming problem involves:
(1) providing the problem's name, letter and description
(2) deciding specific limits (e.g., CPU, memory)
(3) supplying the tests' input and output
(4) decide how much points each test is worth
(5) indicate whether it should be shown to the user

The first and second steps usually involve using a system that either generates HTML or PDF and then uploading the file containing the description to Mooshak through the web interface along with filling the other required fields. The remaining tasks are performed for each test using the web interface. In the current year, the author has used Mooshak in several courses, one of them with around 250 students, and has created around fifty new problems with varying degrees of complexity, some with 100 different test cases. In many of these cases, tests address different problems and thus have to be organized in several categories, with different feedback messages and points. When talking about these numbers, it is very important to have a tool that is able to help create tests in an automatic or semi-automatic manner.

4 Creating a new problem using the tool

The solution proposed here is to use a domain specific language (DSL) for creating problems by using a text editor. In order to create a problem or set of problems, the user needs to create a text file using a simple textual format. The user may specify the following fields:

NAME This is the first keyword given and supplies the name of the problem; the problem ends with the corresponding END keyword
LETTER This is the name of the folder that stores the problem and what is stored in letter
TESTS One or more tests; it ends with an END keyword and may have the following fields:
  INPUT One line of input
  LONGINPUT One or more lines of input; it ends with an END keyword
  INPUTGEN An optional integer n followed by a function (either a lambda function or the name of a function in an imported module) that will be used to generate n inputs
  FEEDBACK A feedback message for these tests
  POINTS How many points to award to each of these tests
  SHOW If given, these tests will be shown
IMPORT The name of a module to import
CODE One or more lines, ending with an END keyword. This creates a module and adds the code into it
SOLVER This may be either a lambda function or the name of a function in an imported module and will be used to create the outputs
DESCRIPTION One or more lines, using the markdown syntax, ended with the END keyword
POINTS If given it evenly distributes this number of points to all tests

Listing 1 shows a contrived example of a problem. The first two tests have feedback and are shown to the user. The following test is completely blind: it neither has any feedback nor is shown to the user. The following 10 tests are generated automatically and have a feedback and are shown. The last five tests are somewhat larger and also have a feedback but are not

shown. The function that solves this problem was supplied by a Python lambda function. The description uses the markdown format supplied by the markdown module [12]. All these tests sum to 100 points that are evenly distributed among them. In this case, all the code is contained in the file instead of importing a module. It is possible to import several modules, one per IMPORT keyword and also supply several CODE keywords.

Listing 1 An example.

```
NAME     Summing a bunch of  numbers
LETTER   A
CODE
def gen_list(size, min_val, max_val):
        import random
        return lambda: ' '.join(str(random.randint(min_val, max_val))
            for i in range(size))
pequeno = gen_list(10, 1, 10)
medio = gen_list(100, 1, 100)
END
SOLVER  lambda s: sum(int(x) ** 2 for x in s.split())
TESTS
        INPUT   10 20 30
        LONGINPUT
12
13
END
        FEEDBACK        Examples given in the problem description
        SHOW
END
TESTS
        INPUT   15 25 35
END
TESTS
        INPUTGEN 10 pequeno
        FEEDBACK small tests
        SHOW
END
TESTS
        INPUTGEN 5 medio
        FEEDBACK larger tests
END
DESCRIPTION
# Sum a list of numbers
Create a program that:
-  reads several numbers and
-  prints the sum of their squares
END
POINTS 100
END
```

A file may have more than one problem. Each problem starts with the NAME keyword and ends with the corresponding END keyword. In order to use the system, one simply has to execute the command `moopec` with the files as arguments:

```
moopec example.txt
```

which will create one folder with the corresponding letter for each problem. Then it is just a matter of copying these folders into the Mooshak's problem directory inside the corresponding contest and adjust the permissions accordingly.

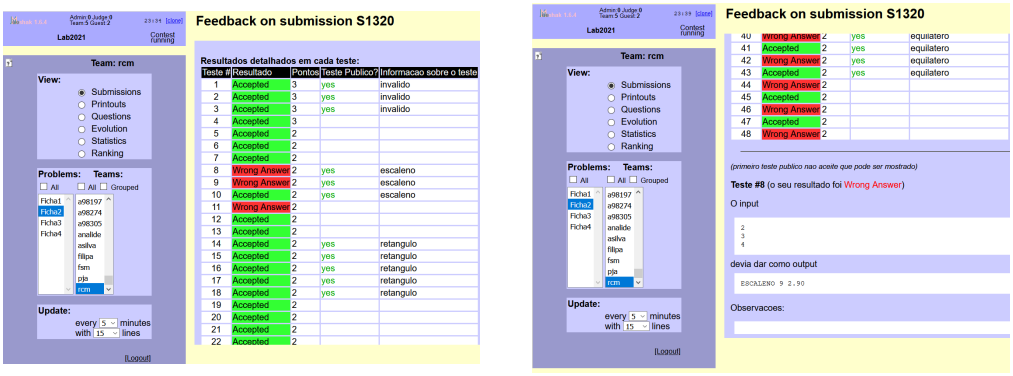

Figure 1 Example of feedback of a problem in Mooshak.

Figure 1 shows an example of the feedback of a problem generated using this system inside Mooshak [1]. This problem has 48 tests and had 1775 submissions. The goal is to find the type, perimeter and area of a triangle given the lengths of its sides.

5 Conclusions

Moopec aims to facilitate the batch creation of problems for the Mooshak system. Since this system has been used to host several ICPC and IOI events for many years, it is robust and can be used extensively for hosting programming assignments. Moopec facilitates the batch creation of programming exercises. It not only allows instructors to handcraft tests but also to use generators that are able to create tests for evaluating specific concerns. Given that Mooshak can be used for many categories of programming assigments and Moopec helps automate the creation of these exercises, it helps instructors be more productive and easily create new assignments, adapt older ones or combine them.

It may seem that one of the vulnerabilities of the system is that the solution can only be implemented in Python, However, since Python can easily call other programs using `os.system` or `subprocess`, this is not a significant drawback. The main advantage of this tool is to simplify the task of creating a problem set into that of creating a text file. While this may not seem to be as intuitive as some of the usability design principles advocate, it greatly speeds up the task of creating problem sets and thus greatly increases the instructor's productivity.

This tool has already been used in practice for creating several kinds of programming assignments, ranging from simple programming exercises to evaluating full-fledged projects. In the current year, the author of this work has used Moopec for creating 9 contests with close to 50 different problems. Many of these problems had around 50 tests, with some having 100 tests and most of them had a message documenting what was the situation addressed by the tests. While some of the tests were handcrafted for testing specific purposes, others used generators for checking specific concerns. The contests in the second semester of 2021 are being used by students with over 8000 submissions and will probably be closer to 9000 until the end of the semester. One of these contests is a project where the users submit a zip file containing C code that is then compiled and run.

[1] The feedback code was gently provided by Pedro Ribeiro.

Even though the feedback could be improved, the current version is already quite useful. It allows partial evaluation, i.e., it uses an IOI grading philosophy and provides feedback that helps users understand which tests failed. In the project assignment mentioned above, all the tests were public and thus the students are able to understand which conditions caused their program to fail and thus try to address the problem. The sheer amount of tests deters students from trying to cheat the system by simply creating code that passes all the tests. While this philosophy has its problems as have other similar ones [3], it is an invaluable tool when trying to create tasks for continuous evaluation of students in an online context without overburdening instructors with the gigantic task of evaluating all these submissions and providing adequate feedback. Future work will add extensions to the DSL in order to provide static and dynamic correctors.

References

1. Matt Bower. A taxonomy of task types in computing. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 281–285, 2008.
2. Codeboard. `https://codeboard.io/`. Accessed: 2021-04-07.
3. Michal Forišek. On the suitability of programming tasks for automated evaluation. *Informatics in education*, 5(1):63–76, 2006.
4. International collegiate programming contest. `https://icpc.global/`. Accessed: 2021-04-07.
5. International olympiads in informatics. `https://ioinformatics.org/`. Accessed: 2021-04-07.
6. Tony Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58. Citeseer, 2002.
7. José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.
8. José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet Another Programming Exercises Interoperability Language (Short Paper). In Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós, editors, *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.SLATE.2020.14`.
9. Dale Parsons and Patricia Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 157–163, 2006.
10. Ricardo Queirós and José Paulo Leal. Babelo—an extensible converter of programming exercises formats. *IEEE Transactions on Learning Technologies*, 6(1):38–45, 2012.
11. Alberto Simões and Ricardo Queirós. On the Nature of Programming Exercises. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 24:1–24:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2020.24`.
12. Manfred Stienstra, Yuri takhteyev, Waylan limberg, and Waylan Limberg. Python-markdown. `https://pypi.org/project/Markdown/`. Accessed: 2021-04-07.
13. Errol Thompson, Andrew Luxton-Reilly, Jacqueline L Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 155–161, 2008.