Olympiads in Informatics, 2024, Vol. 18, 175–184

DOI: 10.15388/ioi.2024.14

IOI Project Report on Improving TPS (Task Preparation System)

Kian MIRJALALI[1], Ali BEHJATI[2]
[1]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
[2]*Douro Labs, Porto, Portugal*
e-mail: mirjalali@ce.sharif.edu, ali@dourolabs.xyz

Abstract. The Task Preparation System (TPS) is primarily designed for preparing IOI tasks. Initially developed and successfully utilized during IOI 2017, it has since been employed in various nationwide and international programming contests, such as IOI 2019~2024. Based on feedback received over the years, the tool required further development to enhance its usability, functionality, and maintainability. This article is the conclusion report of an IOI project defined for a specific set of improvements on TPS.

Keywords: IOI project, competitive programming, task preparation, Olympiad in Informatics, programming contest.

1. Introduction

The host technical and scientific committees of IOI 2017 developed several software tools during their preparations for the contest. One of these tools, the Task Preparation System (TPS), which was specifically created and utilized for preparing the contest tasks, received highly positive feedback from both the HSC and ISC members. The tool is publicly available on GitHub[1] under the MIT License[2] and has been widely used in numerous programming contests, including IOI 2019~2024, Iran's national IOI team selection contests, and the ICPC Regional contests (Tehran site). Due to the extensive use of TPS, we presented its application and shared insights with other members of the IOI community during a talk at the IOI conference in 2019, as well as by publishing an article in the IOI journal (Mirjalali *et al.*, 2019). We recommend reading that article first to become acquainted with the specific details and features of TPS.

[1] `https://github.com/ioi-2017/tps`
[2] `https://github.com/ioi-2017/tps/blob/master/LICENSE.txt`

TPS is a standalone collection of tools primarily written in Python and Bash scripts, specifically designed for preparing the tasks (also known as problems) in programming contests. Typically, TPS operates through a command-line interface and is employed in an offline environment. However, the task/contest directory is often shared with collaborators using version control systems like `git`. The process of preparing a high-quality contest problem is intricate and requires careful handling of multiple steps and components. Some of the key elements involved, but not limited to, include:

- Task statement, usually written in latex or markdown format.
- Designing function signature and IO format.
- Specifying the task constraints such as time/memory limits and restrictions on input values.
- Designing subtasks and their score.
- Graders: programs (written for each programming language allowed in the contest, such as C++ and Java) that link with the contestant's solution and provide it with the grading interface, say for reading the input and writing to the output.
- Task (public) attachment: the set of files provided to the contestants during the contest, such as sample test data, compilation scripts, and basic graders for local testing.
- Input generators and validators.
- Parameters for generating the test data.
- Assignment of test data to subtasks.
- Solutions; including correct, wrong, slow, and suboptimal solutions written for specific subtasks.
- Checker: a program that verifies the output of the contestant's solution per test case and specifies its score.

TPS, being a command-line interface, streamlines the preparation of programming problems by automating various error-prone tasks and effectively detecting errors and warnings. Fig. 1 provides an example of how TPS generates the test data for a task. Moreover, as an open-source project, TPS offers easy customization options, allowing users to tailor it to their specific requirements.

```
> tps gen
generator          compile[WARN]
solution           compile[OK]
validator          compile[OK]
0-01               gen[OK]     val[OK]     sol[OK]
1-01               gen[OK]     val[FAIL]   sol[OK]
1-02               gen[OK]     val[OK]     sol[OK]
2-01               gen[OK]     val[OK]     sol[OK]
2-02               gen[OK]     val[OK]     sol[OK]
3-01               gen[OK]     val[OK]     sol[OK]
3-02               gen[OK]     val[OK]     sol[OK]

Finished.
```

Fig. 1. An example of executing "`tps gen`".

A sign of a software being alive and under usage is the flow of bug reports, suggestions, and feature requests. According to Lehman's laws of software evolution (Lehman *et al.*, 1997):

- A system must be continually adapted and its functional content must be continually increased, or else it becomes progressively less satisfactory over its lifetime.
- As a system evolves, its complexity increases unless work is done to reduce it.
- The quality of a system will decline unless it is rigorously maintained and adapted to operational environment changes.

As the original developers, we have voluntarily maintained TPS since 2017, finding it enjoyable and meaningful work. Based on user feedback, we recognized the need to enhance the usability of TPS's existing features, introduce new functionalities, and ensure its maintainability through code refactorings. However, accomplishing these tasks proved to be time-consuming, highlighting the necessity for dedicated resources to tackle more substantial improvements. After thorough consultations, we decided to propose this further development of TPS as a project supported by the IOI. Our project proposal was accepted, granting us the opportunity of making significant enhancements to TPS. This article serves as a report on the progress made during this IOI project. We will first provide a brief overview of the software state before the project commencement, and then, we will explain the improvements made throughout the project.

2. Software State before the Project

Before the start of this IOI project, TPS had already undergone several changes since 2017. The following is a summary of the improvements made during this period. Please refer to the GitHub history[3] for more details.

- Recurring refactorings to keep the software maintainable.
- Resolved several reported bugs.
- Some updates on the offline markdown viewer tool[4].
- Improvements and bug fixes for Windows users.
- Better error handling, including detection of missing generated tests.
- Improved the compilation process; detecting compile warnings and adding the option for showing verbose details.
- Added Python as a language for solutions.
- Generalized the scripts to handle output-only, two-step, and communication tasks out of the box (without the need to customize the scripts per task).
- Added the testing framework with more than 300 tests for the "`tps`" command.

[3] `https://github.com/ioi-2017/tps/commits/master`
[4] `https://github.com/ioi-2017/markdown-viewer`

- Added multiple configurable settings for tasks: grader name, [not] having checker, [not] availability of {C++, Java, Python, Pascal} as solution languages.
- Added the general "`export`" command to produce a package for importing into contest systems in order to reduce the manual work; specifically, added the exporter script for CMS.

3. TPS Improvements in the IOI Project

An initial list of technical tasks was created as a starting point for the project. However, as is typical in software projects, some tasks were removed from the list after conducting more thorough cost-benefit analyses, while new tasks were added due to circumstances. Although the initial estimation was around 500 hours, it ultimately required over 700 hours to complete all the tasks on the updated list. We now go through the major tasks that were accomplished as part of this project.

Improvements in Software Design and Behavior

Several improvements have been made in the TPS behavior, addressing bugs, handling user-induced errors, and enhancing the user interface. For instance, there are now more informative details available regarding the behavior of a solution when invoked against the provided test data. Furthermore, over 70 refactoring commits have been made throughout the project in order to improve the code quality and maintainability. These refactorings played a crucial role in keeping the codebase clean while developing other features.

Improving the Test Suites and Testing Infrastructure for the TPS Software

Having automated tests is crucial for achieving acceptable software delivery performance in terms of tempo and stability (Forsgren *et al.*, 2018). Without them, making software modifications can gradually become as challenging as walking through a minefield. Throughout this project, nearly 1500 tests have been incorporated in the TPS git repository, encompassing unit tests for common utility functions, tests on "`tps`" command itself, and tests on subcommands such as "`tps gen`" and "`tps invoke`". Additionally, behavioral tests have been added for a modified version of the "`testlib`" header, tailored specifically for CMS and IOI tasks[5].

In order to accomplish this goal, we made more than 45 commits dedicated to improving the testing infrastructure. These changes encompass a range of enhancements, including the addition of tools for probing the state of variables and files after the execution of a command. Fig. 2 provides usage examples to illustrate these improvements. In the first test of this example, it is expected that running the command "`set_variable my_new_var "my new value"`" will set the value of variable "`my_new_var`" to "`my new value`" without printing anything in the standard output/error streams. The second test states that executing the command "`tps gen`" (in a predefined environment)

[5] `https://github.com/ioi-2017/tps/tree/master/extra-assets/testlib`

```
expect_exec -vs my_new_var "my new value"    \
    -oempty -eempty                          \
    set_variable my_new_var "my new value"

expect_exec -f "tests" "td3/probed_files/0_tests"    \
    -o "td3/stdout" -eempty                          \
    tps gen
```

Fig. 2. Examples of probing the state of variables and files in TPS tests.

shall print the contents of file "`td3/stdout`" in the standard output, and nothing in the standard error stream. Furthermore, it should create a directory with name "`tests`" and contents exactly matching the directory "`td3/probed_files/0_tests`". Please refer to the testing documentation[6] for more comprehensive details.

Completing/Updating the Documentation

The official documentation is now up-to-date, thoroughly explaining all features. Moreover, a brief technical documentation has been provided, covering internal code styles, patterns, and conventions. Separate comprehensive documentations have also been added for creating TPS task templates and for testing the TPS software itself. Furthermore, the "`extra-assets`" directory contains appropriate versions of Makefile, gitignore, and the "`testlib`" header specifically tailored for CMS and IOI tasks. These additions aim to provide users with the necessary resources and guidelines to effectively utilize TPS.

Easier Installation Process

An online installer has been successfully implemented and released for TPS. This installer is prominently introduced in the first-page README file of the project. Users can now easily install TPS by executing the following command. This streamlined installation process eliminates the need for manual cloning of the project from GitHub and running the installation script. As a result, the installation process for new TPS users is greatly simplified.

```
bash -c "$(curl -fsSL
    https://raw.githubusercontent.com/ioi-2017/tps/master/online-installer/install.sh)"
```

Extending the Task Exporters

A task exporter is a script for transforming the data of a prepared task into a package with a predefined format suitable for importing into an online judge system. Using task exporters reduces the manual work and automates the error-prone process of adding problems to contest systems. A task exporter for CMS was already implemented in TPS before this IOI project. In order to enhance the usability of TPS, it is crucial to implement exporters for other online judge systems as well. In this regard, we have now implemented the

[6] `https://github.com/ioi-2017/tps/blob/master/tests/README.md`

exporter for DOMjudge, the online contest system primarily used in ICPC. To use the exporter, users can run the command "`tps export DOMjudge`" in the directory of a prepared task. The exporter will then create an archive that can be uploaded to the administration system of DOMjudge. TPS architecture is designed to be flexible, allowing for easy addition of exporter scripts for other online judge systems in the future.

We have also introduced a second protocol for the existing task exporter for CMS. This new protocol aims to enhance the integration between TPS and CMS by providing increased configurability through the TPS directory structure. To export a prepared task for CMS, users shall now run the command "`tps export CMS <protocol-version>`" within the task directory, where the parameter "`<protocol-version>`" can be specified as either "`1`" for the old protocol, or "`2`" for the new protocol.

Adding the Command "stress"

This command is designed to subject a solution to stress testing. Specifically, it executes the solution against a series of randomly generated test cases with the aim of identifying a test case that causes the solution to fail, commonly known as being *hacked*. This tool is particularly helpful in the process of finding tests for distinguishing incorrect solutions. The stress testing procedure is conducted in a series of rounds, with the following steps being carried out in each round:

1. A "test case generation string" is randomly produced; we will later explain how this is done. This string is a single-line text similar to the test generation lines written in the file "`gen/data`".
2. The test case input is generated from the test case generation string, with the same method as the process of generating the task test cases using "`tps gen`".
3. The generated test case input is validated by the input validators.
4. The corresponding test case output is produced by the model solution.
5. The stressed solution is invoked with the generated test case as input. The score and verdict of the invocation is specified through a process similar to the command "`tps invoke`".
6. The stressed solution is considered to be hacked by the generated test case if it does not get the required score.

A sample execution of the command is depicted in Fig. 3. Alongside the information displayed in the terminal output, the test case generation strings which expose faults in the solution, are also recorded in a separate file. This allows for further analysis and examination of the specific test cases that triggered the failure, or using them as the task test data.

The stress command gets two positional arguments. The first argument specifies the path of the solution file to be stressed. The second positional argument is one of the following:

- The path to a test case generation file; a python file which produces the test case generation strings. The python file shall implement a function "`gen_command()`" that returns a test case generation string upon each call. A sample test case generation file is shown in Fig. 4.

```
> tps stress "my-solution.cpp" "gen1 80 {random.randint(1, 70)} {ustr(8, 9)}"
test-gen-file       create[OK]
test-gen-file       verify[OK]
generator           compile[OK]
validator           compile[OK]
model solution      compile[OK]
stressed solution   compile[OK]
checker             compile[OK]
Round 1:
gen1 80 56 KjqdZ77ZT
gen[OK]   val[OK]   model[OK]   stressed[OK]   0.016   check[OK]      1  [Correct]
Round 2:
gen1 80 7 4wjafMy_c
gen[OK]   val[OK]   model[OK]   stressed[OK]   0.017   check[OK]      1  [Correct]
Round 3:
gen1 80 59 0d6Uo8i00
gen[OK]   val[OK]   model[OK]   stressed[OK]   0.018   check[OK]      0  [Wrong Answer]
Hacked!
Round 4:
gen1 80 4 ISFYrwLk4
gen[OK]   val[OK]   model[OK]   stressed[OK]   0.019   check[OK]      1  [Correct]
Round 5:
gen1 80 18 JClqyX58d
gen[OK]   val[OK]   model[OK]   stressed[OK]   0.015   check[OK]      0  [Wrong Answer]
Hacked!
 ⋮
```

Fig. 3. A sample execution of "`tps stress`".

```
from stress_test_gen_utils import *

def gen_command():
    return "gen1 80 {} {}".format(
        random.randint(1, 70),
        ustr(8, 9),
    )
```

Fig. 4. A sample test case generation file for "`tps stress`".

- A test case generation format string; a general string used for producing test case generation strings. The string must be in the shape of a Python *format string* that produces a test case generation string upon each evaluation. Below is the test case generation format string equivalent to the sample test case generation file in Fig. 4.

```
"gen1 80 {random.randint(1, 70)} {ustr(8, 9)}"
```

The second positional argument of the stress command is interpreted as a test case generation file path if an ordinary file exists with the same path as that argument. Otherwise, it will be interpreted as a test case generation format string.

Adding the Command "init"

Creating the TPS directory structure manually for a new task can be error-prone and cumbersome. However, this process has now been automated with the introduction of the command "`tps init`", which is similar to the widely known command "`git init`". By executing "`tps init`", users can simply initialize a new task directory based on a specified task template. Currently, the directory "`task-templates`" in the TPS git repository contains a ready task template named "`default`" which is specifically designed for IOI batch tasks. However, it is also easy for users to create and use their own custom task templates. Comprehensive documentation on task templates is available to provide guidance in this regard[7].

The process of initiating a new task using "`tps init`" generally starts with a user interaction, prompting for a few task template parameters needed for building the correct directory structure. Such an interaction is depicted in Fig. 5 (user inputs are in boldface and blue color for clarity). It initializes a task in a new directory "`day1-book`" using the template "`default`" located at "`tps/task-templates`". Additionally, the option "`-D has_java=false`" defines the variable "`has_java`" as "`false`" and bypasses prompting the user for this variable during the interaction.

```
> tps init "day1-book" -T "tps/task-templates" -t "default" -D has_java=false
Running the instantiation script 'tps/task-templates/default/task-template-instantiate.sh'...
Template parameter 'short_name'...
Enter a value of type 'identifier' for 'short_name':
book
Template parameter 'task_title' (Shown as heading of statement)...
Enter a value of type 'string' for 'task_title':
The Book
Template parameter 'has_grader' (Are solutions linked with graders)...
Enter a value of type 'bool' for 'has_grader':
y
Template parameter 'grader_function_name'...
Enter a value of type 'identifier' for 'grader_function_name':
solve
Template parameter 'has_java' (Is Java language available for solutions)...
Parameter 'has_java' has predefined value 'false'.
Template parameter 'has_public' (Is public data provided to the contestants)...
Enter a value of type 'bool' for 'has_public':
y
Template parameter 'statement_format' (Is statement in markdown or tex format)...
Enter a value among {md, tex, none} for 'statement_format':
md
Copying task template 'tps/task-templates/default' to the new directory 'day1-book'...
Done.
Entering the new directory 'day1-book'
Replacing '__TPARAM_HAS_JAVA__' with 'false' in content of file 'problem.json'...
Done.
Removing files related to language Java
Replacing '__TPARAM_SHORT_NAME__' with 'book' in all file contents under '.'...
 ⋮
The instantiation script execution finished successfully.
Finished. Task directory 'day1-book' is ready.
```

Fig. 5. An example of interacting with "`tps init`".

[7] `https://github.com/ioi-2017/tps/blob/master/docs/task_templates.md`

4. Conclusion

Despite the conclusion of this IOI project, the influx of bug reports, feature requests, and improvements for TPS continues, as is typical for any live project. We hope to engage more collaborators and contributors for the project in the future. Additionally, we are interested in establishing correspondence with programming contest organizers to showcase TPS, offer assistance in its usage, and gather feedback.

Acknowledgments

The members of the development team for this IOI project were Kian Mirjalali, Ali Behjati, Peyman Jabbarzade, and Mahdi Shokri. We would like to thank Amir Keivan Mohtashami, Amir Mohammad Dehghan, Ali Sharifi Zarchi, Mohammad Ali Abam, Hamid Zarrabi-Zadeh, and ISC members, especially Jonathan Irvin Gunawan for their useful comments on this project. We should also acknowledge Farid Ahmadov (Azerbaijan), Ali Sharifi Zarchi (Iran), Mohammad Mahdian (Iran), Mohammad Ali Abam (Iran), Kresimir Malnar (Croatia), Madhavan Mukund (India), Eslam Wageed (Egypt), Musa Alrefaya (Palestine), and Haris Gavranovic (Bosnia and Herzegovina) for their kind testimonials and endorsements for this IOI project.

References

Forsgren, N., Humble, J., Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press.

Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M. (1997). Metrics and laws of software evolution-the nineties view. In: *Proc. 4th International Software Metrics Symposium* (*METRICS '97*). pp. 20–32.

Mirjalali, K., Mohtashami, A.K., Roghani, M., Zarrabi-Zadeh, H. (2019). TPS (task preparation system): A tool for developing tasks in programming contests. *Olympiads in Informatics*, 13, 209–215.

K. Mirjalali is a software engineer with a PhD in Computer Engineering Department from Sharif University of Technology. He was a member of the International Technical Committee (ITC) in IOI 2015 and also a member of the Host Technical and Scientific Committees (HTC, HSC) in IOI 2017. He was also an invited HSC member for IOI 2019~2022, 2024. He won a silver medal in CEOI 2003 and became a world-finalist in ICPC 2007. He has been a scientific committee member of Iranian National Olympiad in Informatics (INOI) since 2003, and ICPC in the west Asia region Tehran site since 2009.

A. Behjati is a Software Engineer at Douro Labs. He was a gold medalist in IOI 2015 and 2016. He also has been awarded a bronze medal in ICPC 2019. He was Iran's team deputy leader in IOI 2017 and an invited HSC member in IOI 2020.