Interactive Problem Solving in the Classroom: Experiences with Turing Arena Light in Competitive Programming Education

Giorgio Audrito, Luigi Laura, Alessio Orlandi,
Dario Ostuni, Romeo Rizzi, and Luca Versari

19th IOI Conference, Sucre (BO)
Wednesday, July 30, 2025

Introduction

Turing Arena light

What if we want to use a competitive programming judging system as a teaching tool?

We can use already existing systems:

- *CMS*, *DOMjudge*: hard to set up, not trivial to write problems
- *Codeforces*, *Kattis*: rigid problem format, cannot host privately
- *Terry*: setup made for contests, cannot have interactivity

Goals

The key goals for such a system are:

- *Easy to set up*
- *Easy to write problems*
- *Easy for students to use*
- *Flexible problem format*
- *Unlimited feedback capability*
- *Extensible to new features*

Turing Arena

A first attempt was made with *Turing Arena*.

It was started in 2019 and had much bigger goals.

However, it was too complex and hard to maintain.

Development completely ceased in 2022.

A new requirement was needed: *keep it simple*.

Turing Arena light

Turing Arena light was started as the spiritual successor.

The rationale was to try to achieve the same goals, while keeping it as simple as possible.

This was done both in the design and in the implementation of the system.

Turing Arena light Design I

The fundamental idea behind *Turing Arena light* is to just have two programs that talk to each other: *manager* and *solution*.

The *manager* and the *solution* communicate through standard input and output.

The *manager* is responsible for writing the input for the problem, then reading and scoring the output of the solution.

Turing Arena light Design II

The *solution*, just like in any other judging system, reads the input of the problem, and outputs the solution.

Thus, a problem is defined by a *manager*, plus some metadata.

Turing Arena light is an infrastructure to make this communication easy to setup.

Turing Arena light Design III

This kind of paradigm is already implemented in judging systems, such as *CMS*. **However**, we want to be able to decide where the *solution* and *manager* are run.

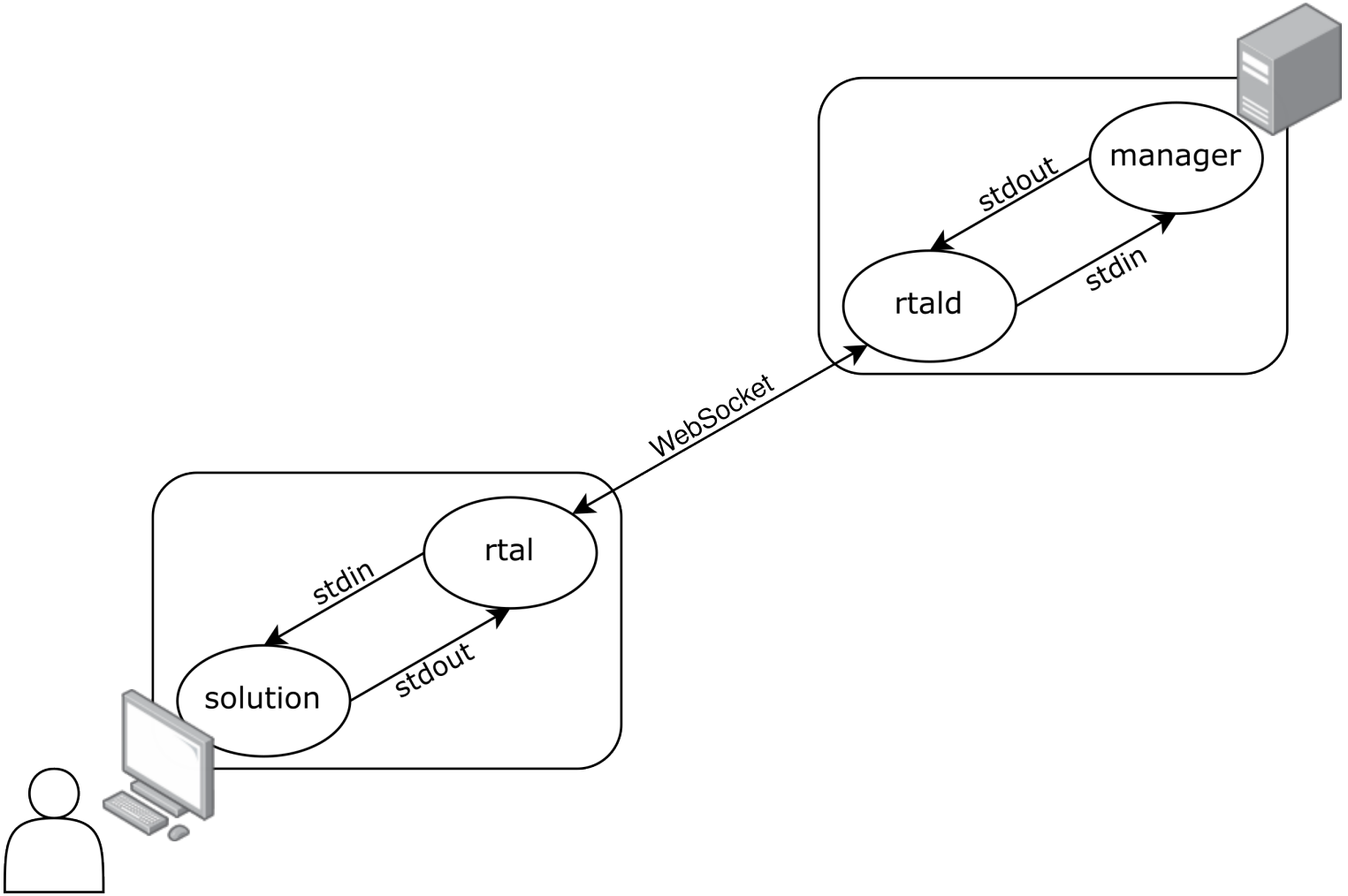
One limit of systems such as *CMS* in a teaching context is that the *solution* is always run on the server, which makes it much harder to debug.

Turing Arena light Design IV

Possibly, we would like to be able to run the *solution* locally on the client, like it happens in *Terry*. However, one limit of *Terry* is that we would lose the ability to have interactive problems.

Turing Arena light brings these two paradigms together. It does so with two components, one client-side and one server-side, attached to the *solution* and *manager* respectively.

Turing Arena light Design V



Turing Arena light Implementation I

In *Turing Arena light*, problems are implemented through *services*. A *service* defines:

- the *manager* program (also called *evaluator*);
- the *parameters* for the *manager* (strings or files);

This information is contained inside a `meta.yaml` file, along with an attachments directory, which usually contains the statement of the problem.

Turing Arena light Implementation II

An example of a `meta.yaml` file:

meta.yaml

```
public_folder: public
services:
    solve:
        evaluator: [python, manager.py]
        args:
            size:
                regex: ^(small|big)$
                default: big
        files:
            - source
```

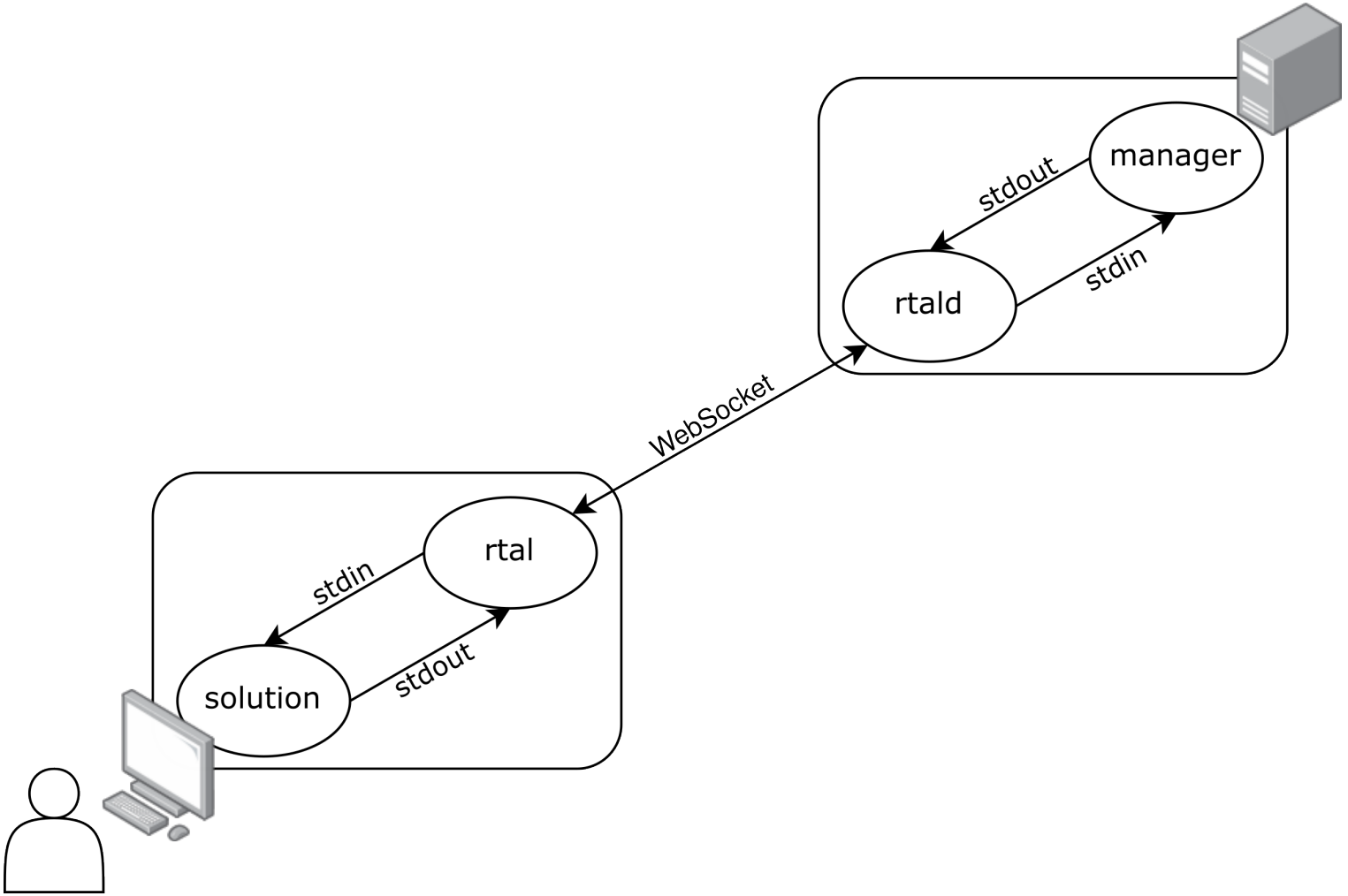
Turing Arena light Implementation III

Turing Arena light is implemented as two components written in *Rust*:

- `rtal`: it's the client component, it connects to the server, runs the solution and mediates the communication;
- `rtald`: it's the server component, given a directory of problems serves them, spawns the appropriate *manager* when asked and mediates the communication.

All network communication is done through *WebSocket*.

Turing Arena light Implementation IV



Turing Arena light Implementation V

Both components have a command-line interface.
Starting the server can be done with:

```
$ rtald -b 0.0.0.0 -d problems
```

Running a solution can be done with:

```
$ rtal -s ws://address connect problem -- ./solution
```

Turing Arena light Implementation VI

`rtal`, the implementation of *Turing Arena light*, only defines the communication protocol, but says nothing about the problem format. Thus, it only defines the *core* of Turing Arena light.

While this allows maximum flexibility, it also means a lot of boilerplate code is needed to write a problem.

Thus, `rtal` allows for *manager* libraries to be written.

Turing Arena light Implementation VII

TC.py and *TC.rs* were written to implement a *Terry*-like problem format for *Python* and *Rust* managers, respectively.

Other than allowing to write a problem in few lines of code, they also support authentication, standardized scoring and saving of the results in a database.

Note that the **full implementation** of `TC.py` is only **66 SLOCs** and `TC.rs` is only **144 SLOCs**.

Turing Arena light Implementation VIII

The size in *Single Lines of Code* of various judging systems:

Judge System	**SLOCs**
CMS	67,147
DOMjudge	94,542
Terry	15,766
Turing Arena	54,421
rtal v0.1	978
rtal v0.2	2,202

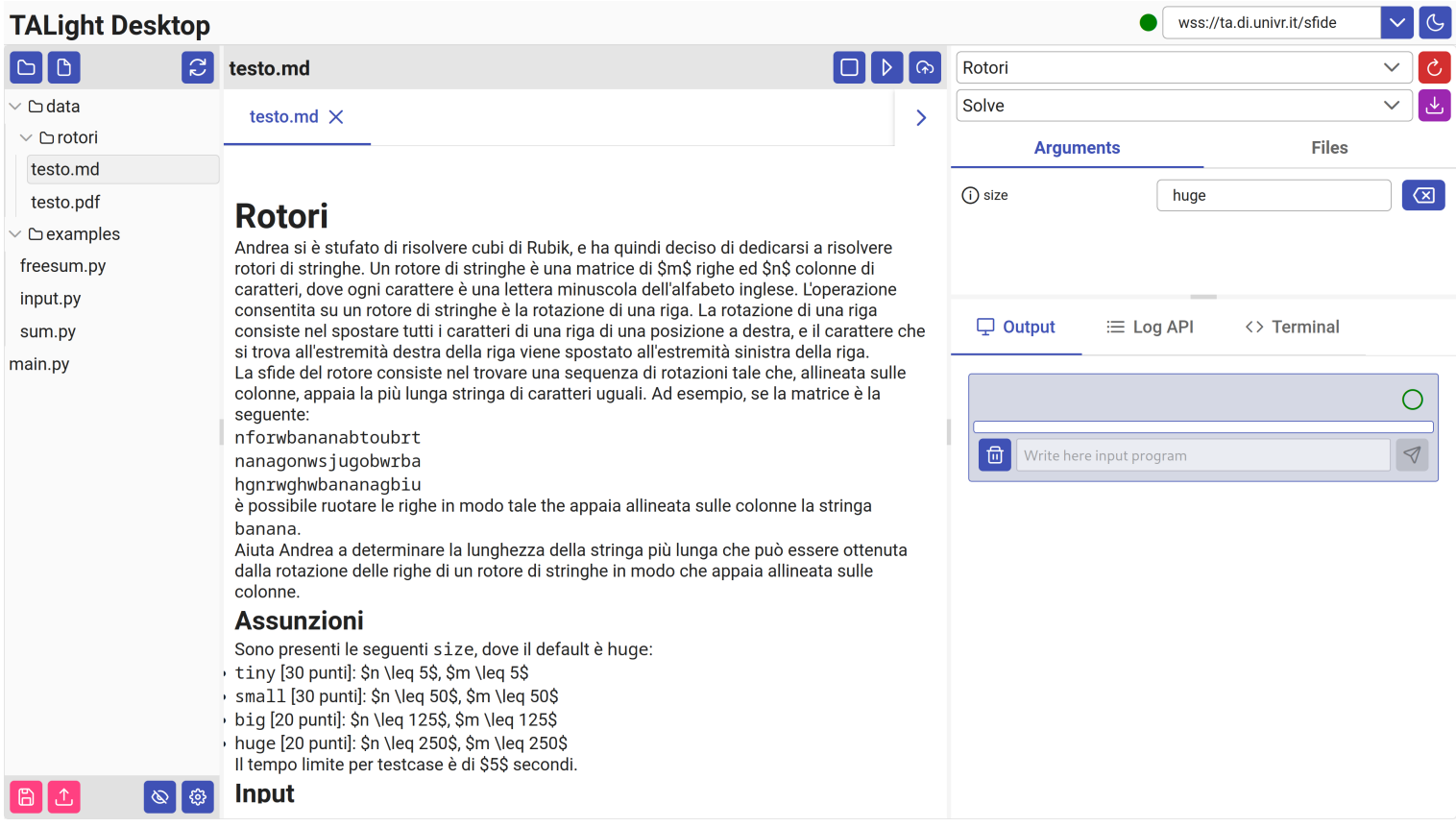
Turing Arena light Implementation IX

While the `rtal` client is a command-line tool, a graphical interface was also developed by the university community.

Since `rtal` only defines the communication protocol, anyone can implement a client for it.

TAlight Desktop is a graphical client that runs in the browser and is able to run *Python solutions* directly in the browser.

Turing Arena light Implementation X



Turing Arena light Experience I

Turing Arena light has been used in several courses at the University of Trento, including:

- *Algorithms and Data Structures* (Bachelor's degree)
- *Competitive Programming* (Master's degree)
- *Advanced Programming* (Bachelor's degree)

It has been used by more than 500 students in total.

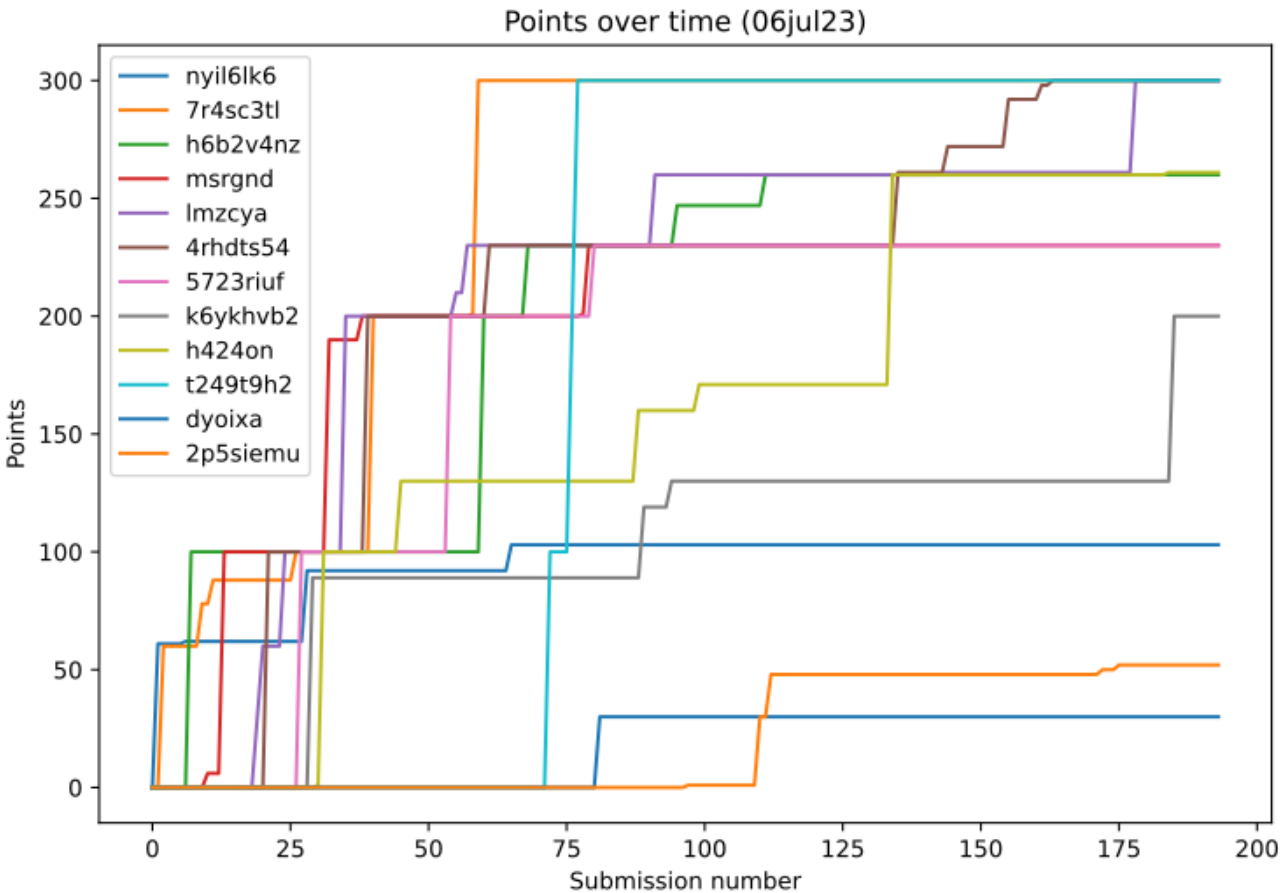
Turing Arena light Experience II

Turing Arena light has been used in the *Sfide di Programmazione* course at the University of Verona for both lab exercises and exams.

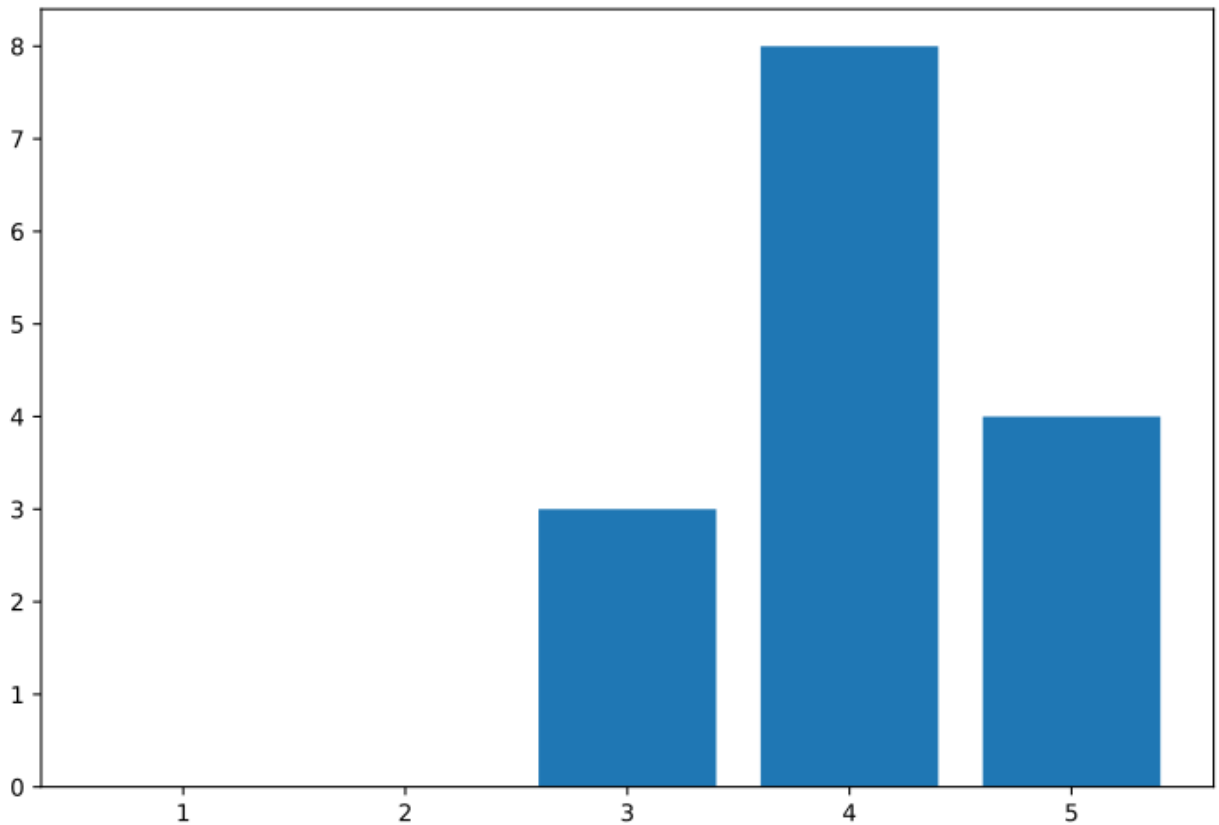
While being used there we built a repository of over 20 problems and we collected feedback from the students.

It has been subsequently used in the *Fondamenti di Algoritmi, Complessità e Problem Solving* course at the University of Verona, with a repository of over 140 problems.

Turing Arena light Experience III

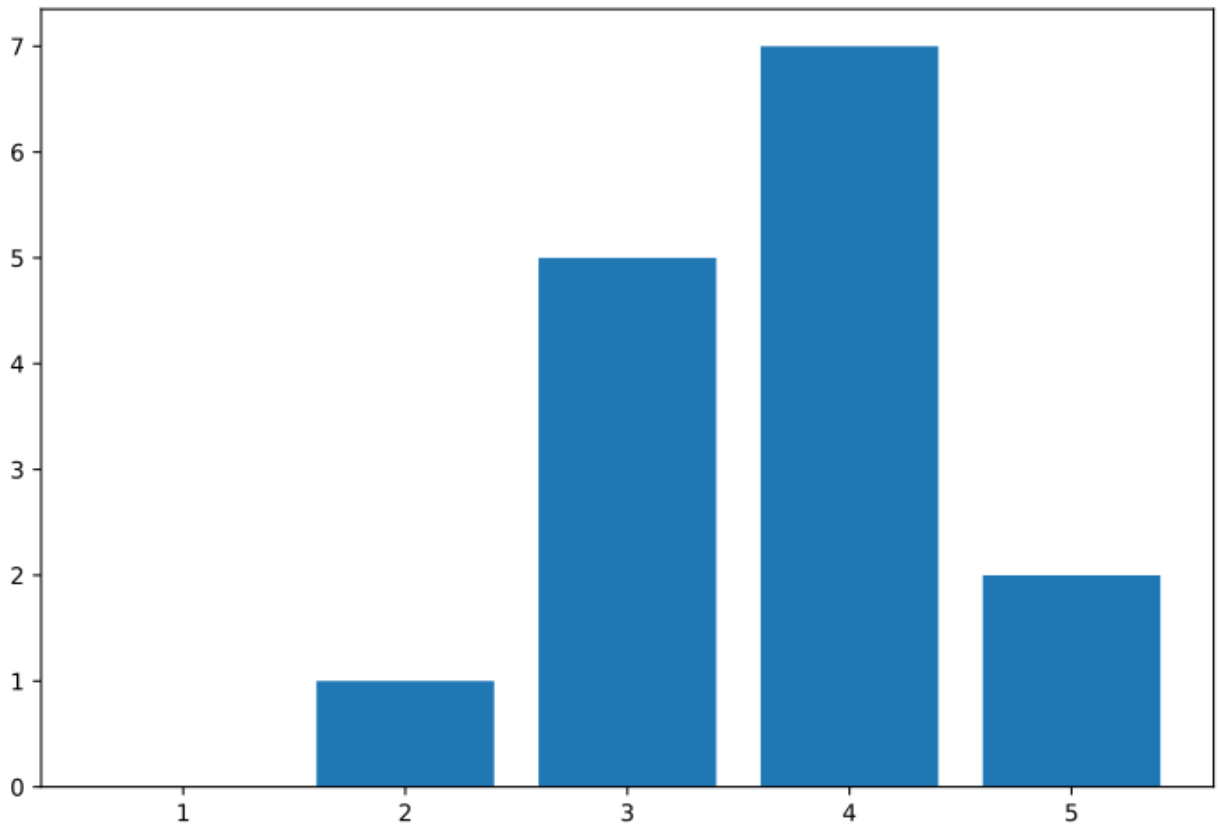


Turing Arena light Survey I



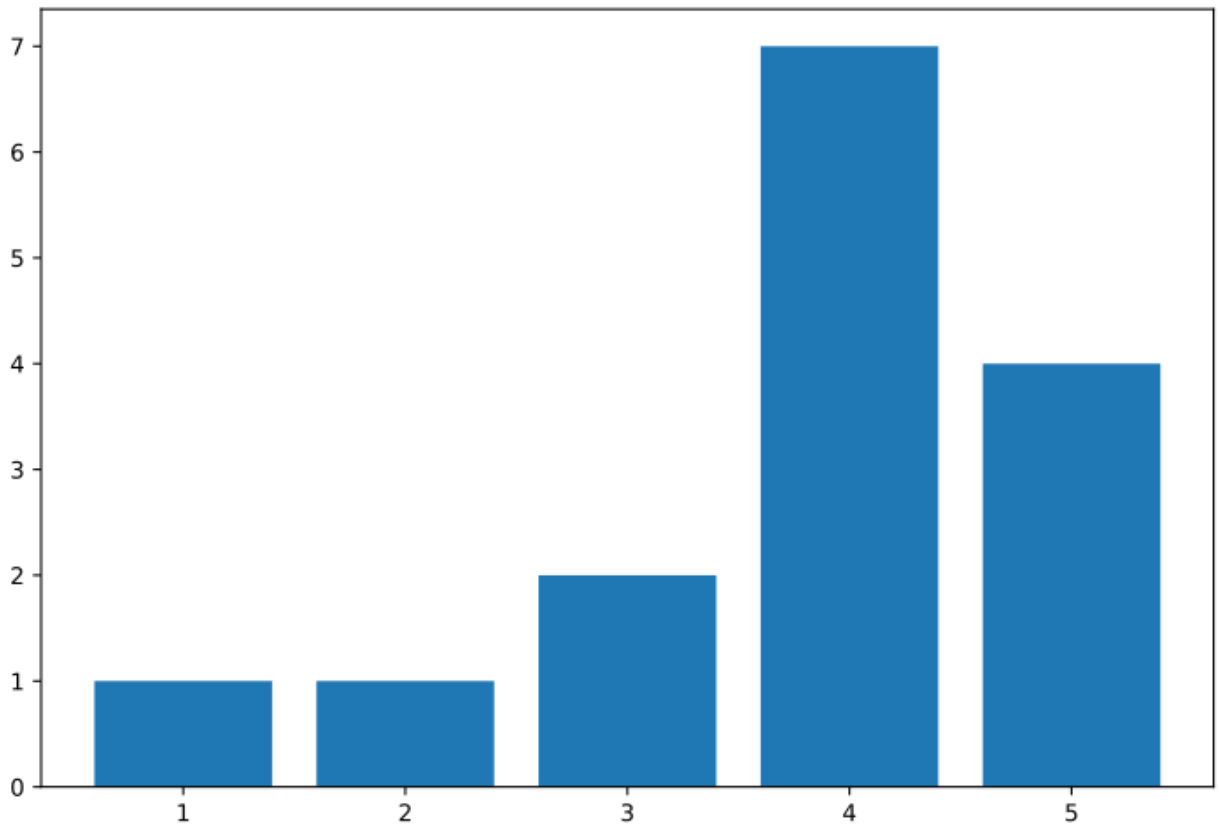
How much did you like the problems available in Turing Arena light?

Turing Arena light Survey II



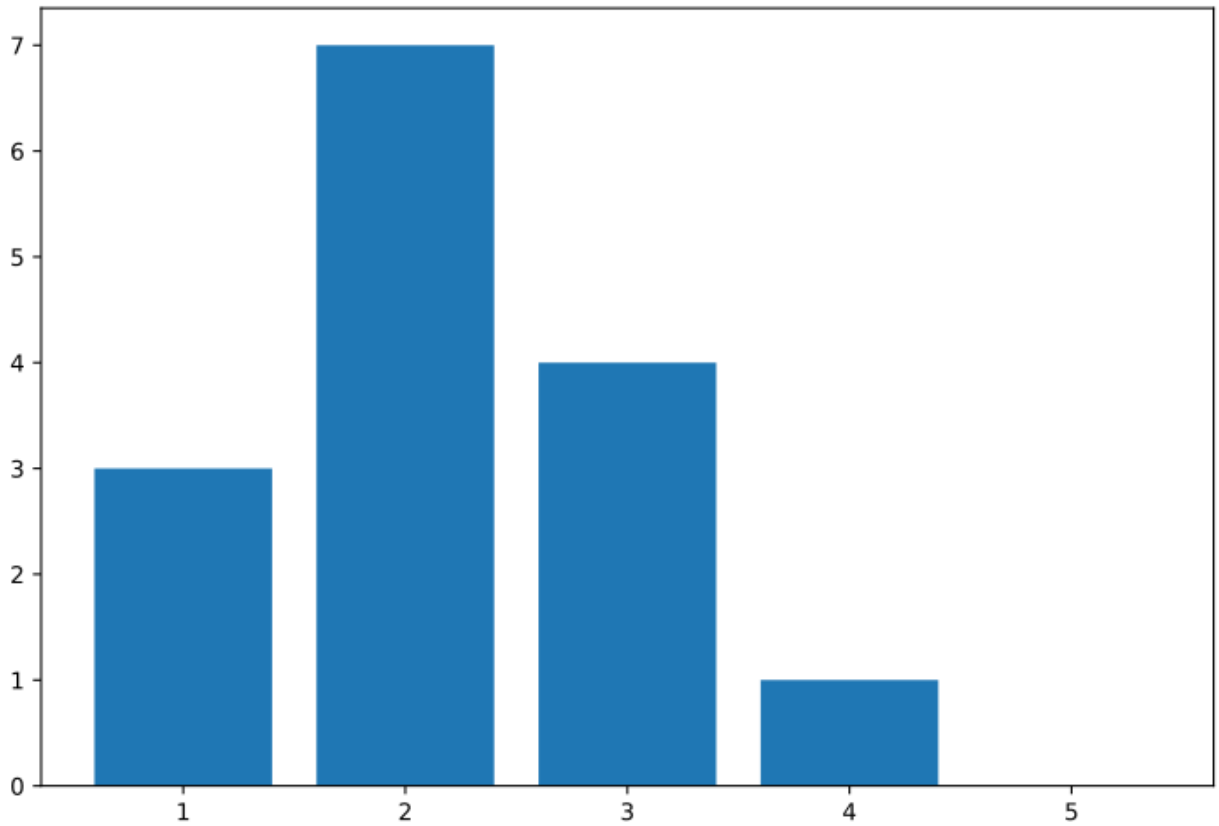
How difficult did you find the problems proposed with Turing Arena light?

Turing Arena light Survey III



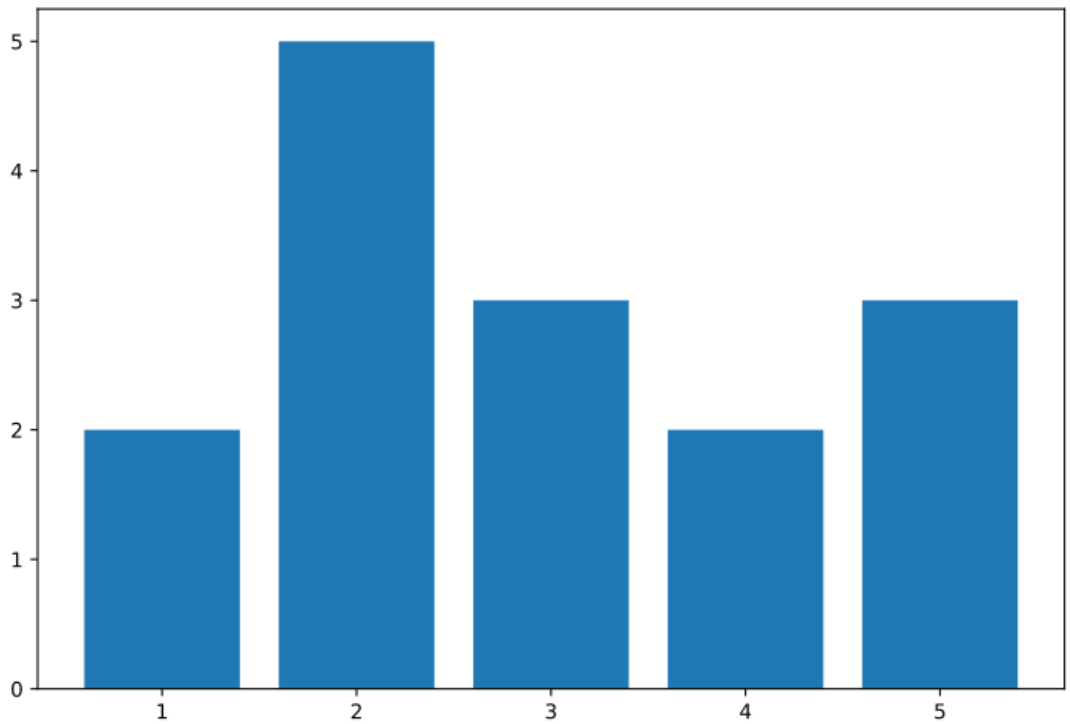
Did you find the interactive problems more interesting than the regular ones?

Turing Arena light Survey IV



How hard was to use Turing Arena light (rtal)?

Turing Arena light Survey V



How strongly would you like for Turing Arena light to have a graphical user interface?

Conclusion

- On the technical side, while flexibility is good for experimenting, some features would need some standardization to move forward, such as time measurement.
- As for the user experience, the feedback was quite positive, the students were engaged and the problems were well received.
- As of now, `rtal` has been forked and new features are being developed by the community, both teachers and students.

Conclusion

- On the technical side, while flexibility is good for experimenting, some features would need some standardization to move forward, such as time measurement.
- As for the user experience, the feedback was quite positive, the students were engaged and the problems were well received.
- As of now, `rtal` has been forked and new features are being developed by the community, both teachers and students.

Conclusion

- On the technical side, while flexibility is good for experimenting, some features would need some standardization to move forward, such as time measurement.
- As for the user experience, the feedback was quite positive, the students were engaged and the problems were well received.
- As of now, `rtal` has been forked and new features are being developed by the community, both teachers and students.