Olympiads in Informatics, 2021 Vol. 15, 3–12

DOI: 10.15388/ioi.2021.01

The Italian Job: Moving (Massively) Online a National Olympiad

Giorgio AUDRITO[1], William DI LUIGI[2], Luigi LAURA[2,3], Edoardo MORASSUTTO[2], Dario OSTUNI[4]

[1]*Department of Computer Science, University of Torino, Italy*
[2]*Associazione Italiana per l'Informatica ed il Calcolo Automatico*
[3]*Uninettuno University, Rome, Italy*
[4]*Department of Computer Science, University of Verona, Italy*
e-mail: giorgio.audrito@unito.it, williamdiluigi@gmail.com, edoardo.morassutto@gmail.com, luigi.laura@uninettunouniversity.net, dario.ostuni@univr.it

Abstract. The COVID-19 pandemic is having a pervasive effect worldwide, including local, national and international Olympiads in Informatics. Most national Olympiads had to be moved online, a process which poses a number of serious challenges. Help across countries is of uttermost importance in this context, to enable a successful continuation of the IOI during globally hard times. In this paper, we share the experience gained and tools produced during a year of online Olympiads in Italy, hoping that other countries can take profit of these (freely available) tools and suggestions for their own Olympiads.

Keywords: team work, programming contest, Olympiads in Informatics, peer education, programming training.

1. Introduction

The *Olimpiadi Italiane di Informatica (OII)*, or Italian Olympiads in Informatics (Casadei *et al.*, 2007), are laid out in phases spanning a two-year length: the **Phase-1** of the N-th edition takes place in November–December of the year $N-1$, the **Phase-2** takes place in April of the year N, and finally the **Phase-3** takes place in September of the year N. The highest ranked students in this phase are then selected for participation in training camps during the whole scholastic year (**Phase-4**), ultimately leading to the selection of the Italian team at the IOI for year $N+1$. The selection process starts with a scholastic pen-and-paper test involving between 10k and 15k participants in Phase-1, which are then reduced to about 1k-2k participants for a regional programming competition in Phase-2. About one hundred of those students are selected for the national programming contest in Phase-3, and 20-30 of them are allowed to enter the training camps in Phase-4.

As a result of the COVID-19 pandemic, all four phases of the OII were affected and thus had to be moved to a fully-online setting. The Phase-1 of the OII 2020 (which happened at the end of November 2019) was the last onsite competition. After that, the Phase-4 of the 2019 edition, the Phase-2, Phase-3 and Phase-4 of the 2020 edition, and the Phase-1 and Phase-2 of the 2021 edition had all to be moved online, after being postponed by a few months in the hope that schools would reopen later in the year. This forced shifting required the development of new tools and solutions, which ensured an overall successful year of Olympiads in Italy. In this paper, we share these (freely available) tools and solution, for the benefit of other countries which may need it for their own upcoming Olympiads. The next three sections present our experience with the first three phases of the OII, in order. Phase-4 has not been included, since for the limited number of participants, it did not require the development of innovative solutions.

2. Phase-1 (Scholastic)

In this phase, students have to answer a "quiz-based" exam, with multiple-choice questions and (numeric) open questions of a logical or algorithmic nature. This contest is normally held in about 500 high-schools, each of them with a designated "contact person" in charge of: distributing the paper-based exam to the students, proctoring them during the exam, collecting the completed exams when the time is up, and finally computing the score of each student. The large number of participants and the "standardized-test" style of the contest made this phase a significant challenge in moving it to an online setting.

Going online. We evaluated several possible solutions to hold this phase online. We initially wanted to restrict as much as possible any possibility of cheating, so we considered using some kind of online platform that would present the questions in a random order and with a fixed time to answer each question (e.g. five minutes) while preventing to "go back" so as to discourage students from sharing answers, since they would have limited time to answer them.

In the end, we concluded that implementing this idea was going to be a significant challenge considering the available time and workforce, and the high requirements in terms of the overall load on the system. We also decided that we didn't want to disrupt the experience so much for the students by putting pressure on them to answer questions quickly and by making them unable to change their answers.

The strategy we finally went with was to stick as much as possible to the "paper-like" feel of the exam, by providing them with a simple PDF file, so the students could immediately see every question and choose in which order to solve them. However, we not only randomized the order of the questions in the file, but we also put randomized data in the actual exercises. The idea behind this was to trick cheating students into suggesting each other wrong answers (since the questions would look almost the same, but would have slightly different data) or at least spend a significant time decoding which "version" of a question they had.

Finally, to ensure that we didn't have downtime caused by the load of tens of thousands of users making requests to our servers at the same time, we offloaded everything we could to external services, like **Twitter** and **Google Forms**.

Randomization of the questions. In order to prepare questions with fully or partially randomized data, we developed a small program that we called **randomTeX** [6]. This program uses Jinja2 as the template engine (with a few configuration tweaks that were necessary to resolve some language ambiguities that came up because LaTeX and Jinja2 have some common syntax, like the opening and closing curly brackets). To feed the data to the template, randomTeX uses an approach similar to the YAML front-matter in Markdown documents.[1]

We prepared a separate LaTeX file for each of the 20 exercises in the exam, and the variations of each exercise were defined by the data in the YAML front-matter of the file itself. This approach turned out to be very flexible, since we could easily add new versions of the same exercise by changing only the exercise file very slightly and without having to touch the LaTeX code.

After some tests, we settled on generating four variations of each exercise: although we could easily have much more than four, we deemed this number enough for our needs and it ultimately made it possible for us to manually verify that each and every variation made sense. We also ended up avoiding multiple-choice questions in favour of numeric open questions, to simplify both the correction process and to minimise the information available to cheating students.

This, combined with a simple randomization in the presentation order of the questions, made it possible to generate *a different PDF for every student*. In the first page of the PDF file we provided information such as the "exam ID" and the First and Last name of the student, to make it clear that the exams were different.

Distribution of the exams. In order to facilitate distribution of the exams (while avoiding a high load on our servers) we decided to simply start distributing the PDF files well ahead of time: we provided a download link 6 hours in advance. Each PDF file, although with different content, was protected by the same password. To disclose the password to every student at the exact starting time of the contest, we used the "scheduled tweet" functionality on Twitter.

Specifically: each student downloaded their custom PDF file ahead of time and, as the start of the exam approached, he or she simply visited the Twitter account of the OII and started monitoring the page for updates. As soon as the contest started, the account automatically tweeted the password that unlocked all the different PDFs.

Collection of answers. We offloaded the problem of collecting answers to an external service: Google Forms. We created a form that was purposefully very simple: a field for the "exam ID" and 20 questions numbered from 1 to 20 without any question detail (as the order was anyway different across students) and with an open numeric field to speci-

[1] For further details, see `https://jekyllrb.com/docs/front-matter`

fy the answer. The idea was to collect the answers in the most reliable possible way, and then perform the actual validation once the contest ended. One detail worth mentioning is that we generated the "exam ID" in a way that it would be easy to reconstruct in case the student would mistype it: we simply concatenated five random words.

To reduce the possibility of last-minute connection issues that students might have had, we specified in the instructions in the first page of the PDFs that it was recommended to submit frequently: this was possible because we enabled "Edit after submit" in the form's settings.

The plan worked well for most of the contest. Unfortunately, several students didn't read or simply didn't follow our advice, and submitted right at the end of the contest: this probably triggered some malicious activity alert (the form started to ask students to solve a CAPTCHA before submitting) and this caused some students to not be able to submit their answers in time. To address this, we had to re-open the submission window (we actually duplicated the Google Form and linked it from the previous form) specifically for "late submissions" and we informed students to use the new link if they weren't able to submit their answers in time, while also reminding them that we reserved the right to accept or discard those late submissions.

Evaluating the submissions was straightforward: when we generated the PDF files we stored the list of correct answers for each file. After the contest we downloaded the spreadsheet with the answers and using the "exam ID" we compared, for each student, their answers with the correct ones. For each exercise we graded only the last submission that included that exercise (since there were no penalties for wrong answers).

Possible improvements. In case we will have to go online again next year, there are a few things that we could improve:

- Instead of using a single Google Form for all the participants, it might be better to use multiple Google Form instances (e.g., ten different forms). We can easily modify the PDF template of the exam so that each student will see in his/her exam PDF a differ*ent link* to the submission form. This might reduce the likelihood of triggering the CAPTCHA requests.
- Upon performing a statistical analysis of correct vs wrong answers for each variation of each exercise, we noticed that in one of the four variation of one specific exercise there was a significantly higher rate of correct answers. This turned out to be caused by the fact that, when we slightly changed the numbers, we introduced an "easier optimal strategy" for the exercise resolution, which was suboptimal in the other variations. Although this didn't greatly affect the results, we were very concerned by this and we are considering introducing some kind of automated validation step in the randomTeX program (e.g. by writing a validation script for each exercise which takes the YAML front-matter as input and outputs a Boolean value) which could help us verify that some conditions are true for all variants of an exercise (e.g. the optimal solution being unique, and so on).

3. Phase-2 (Regional)

In this phase, students have to complete a programming contest, grouped in *territorial* zones based on the geographical position of their school. Italy is a country divided in 20 regions, with very different populations and therefore number of schools. Since this contest was onsite, this heterogeneity used to be an issue: participants from a same region ranged from hundreds to below a dozen (e.g., from 7 to 210 participants in the last onsite contest). In order to mitigate this problem, regions had to be split into *venues* of roughly the same size (from 7 to 51 in the last onsite contest), according to the capacity of the school hosting the contest.

Unlike Phase-1, which mostly evaluates logical and code-reading skills in order to reach as many students as possible, Phase-2 is focused on selecting students that can write code to solve actual programming tasks, both theoretically and practically by writing a program that given an input file produces the correct output. Time and space complexity is not an explicit focus of this phase, as opposed to IOI-like competitions, so every problem is output-only and no explicit time limits are given.

In order to evaluate effectively these skills, the Phase-2 does *not* use the same judge system used in the IOI, but rather one that we specifically designed for this back in 2017, called **Terry** [9]. After accessing the Terry interface, the participants can choose a task and download an input file (which is unique for each student, and changes every time a submission is attempted), run their program locally with the downloaded file as input, and finally upload the produced output file along with the source code of the program producing it (for plagiarism avoidance) and immediately receive a score. This approach was heavily inspired from the early format of the Google Code Jam competition, and allows us to grade submissions of a very large number of students with relatively few resources (as we don't need to run the students' code, except in selected cases) and allows the students to use a broad set of supported languages.

After the contest ends, we perform plagiarism checks. We used to do them using JPlag (Prechelt and Phlippsen, 2000), which however only supports a subset of the allowed languages. For this reason we recently developed Starplag [7], that computes the similarity between two source files regardless of their programming language, by implementing a variation of the Levenshtein distance where text substitutions (i.e., variable renaming) is considered.

Going oline. This was the first phase to be moved online, since it was scheduled to take place in April 2020, right after the March COVID-19 restrictions took place in Italy. Terry was not meant to work as an online judge: instead, it was designed to be deployed "offline" in each of the onsite venues. Since every venue had a designated reference person, we could easily communicate with the students through this person: if a student had a question on the tasks, the reference person (e.g. a teacher in the venue's school) would forward it to us, and we would then provide an answer which would be finally relayed to the student.

Since this process does not work for an online contest, we had to write a new component for Terry handling questions, answers and announcements. This new com-

ponent worked well enough, although leaving margins for future improvements. In particular, it showed all the questions in a "stream" as they arrived to us, making it difficult to reconstruct the context of a question. We underestimated the importance of a "conversation" with a student: a question from a user would often implicitly reference a previous question that the user already sent us, and sometimes a question just couldn't simply be replied with an answer and we had to "answer" the question with another question. For the next year, we plan to develop a conversation-like interface addressing this issue.

Proctoring in real-time more than a thousand students from home with available resources was unfeasible, so we instead adopted the standard approach of online contests, where only after-contest checks are performed. After the contest was over and before extracting the ranking, we examined the submitted source files looking for evidence of plagiarism or irregularities.

Technical aspects. The contest server was designed to run inside a virtual machine sent to the venue hosts, in order to require minimal technical knowledge from them. The virtual machine only had to serve that venue (around 50 students) and be as light as possible since we had no control over the quality of those servers. For this reason, Terry uses SQLite as DBMS: it is light and fast enough for our needs. Scaling a system designed to handle few dozen users to support few thousands users can be risky. To limit this risk, we decided to keep each Italian region in a separate shard of the system, with its own independent instance of Terry. This way each instance only had to process a fixed fraction of the users, also allowing us to migrate each region independently in case of failure of some instance.

We used Docker containers for shipping the replicas and docker-compose for orchestrating them. Ideally, each instance should have used its own host for full separation of the load; however, we ended up renting 8 VMs on Google Cloud Platform[2], assigning each of the 20 regions to a specific VM in a way designed to distribute the load uniformly. Besides the 8 VMs, we also set up a *coordinator* VM to host the communication component and some utility scripts. Among these extra tools, we also had an instance of Grafana[3] displaying many metrics collected by Prometheus[4]: at [4] and [3] you can find a snapshot of our dashboards. From those dashboards you can see the machines we had (named with Greek letters), where *phi* is the coordinator. The communication platform was able to handle an average of 60 requests per second, while the other VMs had an utilization close to zero for most of the contest duration. Unfortunately, due to a misconfiguration of nginx (a wrong request rate limit) the expected spike at the beginning of the contest caused nginx to terminate connections, not allowing them to reach Terry's backend (see the "503" tab in the dashboard [4]). This was fixed in about half an hour; for the future, it will be important to stress test the system also with reasonable loads, not only with *denial of service* rate of requests.

[2] `https://cloud.google.com`
[3] `https://grafana.com`
[4] `https://prometheus.io`

Since the users were partitioned into different servers, we had to make sure that each user connected and logged in the correct container. This was accomplished through a DNS pointing each user to the correct VM, then to the correct instance of Terry within that VM. More precisely, we had separate URLs for each region (e.g., `lom.territoriali.olinfo.it` for Lombardy), adding a `CNAME` record to this domain pointing to one of the VMs, for example `alpha.territoriali.olinfo.it`, that in turn is an `A` record pointing to the VM's public IP. This extra step allowed us to easily move one container into a different VM by simply spawning it and changing the content of the `CNAME` record (a low TTL was set to support this). Fortunately, this measure wasn't necessary as the system run smoothly after the rough start.

4. Phase-3 (National)

Since 2011, the Italian national phase has used the Contest Management System (CMS) (Maggiolo and Mascellani, 2012) as its platform to host the contest; CMS is also the core of the training platform available for students (di Luigi *et al.*, 2016) and has been used also in team Olympiads (Amaroli *et al.*, 2018). The contest is IOI-like: 5 hours and (usually) 3 problems to solve. The only major differences lie in the difficulty level of the problems, which are usually easier than IOI ones (di Luigi *et al.*, 2018), and the set of allowed programming languages: in the Italian national phase only C and C++ are allowed. Before the pandemic, this phase used to be a 3-days onsite event, hosted by a different high school or educational institution each year. The about 100 students admitted to this phase would travel to the contest site with their regional contact person. After the contest, the top 20-*ish* students would continue their journey to the training camps, held in the Italian city of *Volterra*, in *Tuscany*, during which the Italian team for the IOI would be selected.

Going online. As the contest moved entirely online, we had to deal with several problems, the major one being proctoring. While during the previous phases the problem of potential cheating was relatively irrelevant and could be dealt with offline tools, with only about 100 contestants the potential problem of cheating had to be dealt with properly. Thus, much of the organization of the online contest revolved around this point. The competition itself was still hosted with CMS, as its use onsite or online is not significantly different. We let the students use their own machines to compete in the contest and set up a 3-layered proctoring system:

1. All contestants where assigned to one of 14 different Zoom[5] meetings where they would have to be in for the whole duration of the contest, with the camera on and the microphone unmuted. A human proctor from the staff were assigned to each of them, having to look over approximately 8 contestants, while also helping as a communication facilitator.

[5] `https://zoom.us`

2. All contestants were required to compete from inside a virtual machine provided by the organization, having it full-screen for the whole duration of the contest. The virtual machine, aside from blocking access to internet, also recorded the contestant's screen and sent the resulting stream to the server.
3. All contestants were required to run a custom-made proctoring program on the host machine, called *oii-proctor* [5], which would check for misbehaviors (such as opening a browser or leaving the virtual machine) and report them.

Furthermore, in order to mitigate possible unexpected cheating behaviours, the difficulty and novelty of the problems was increased relative to previous years, and the number of people selected for the training camps was also increased to 29. This was also made possible by the simpler logistic and lower budget required by an online training camp, which used to be limiting factors for this number.

The handling of the contest was overall successful, with only some secondary issues:

- Contestants with bad internet connections or very slow PCs had, inevitably, a disadvantage.
- The increased difficulty of the problems almost halved the average score: in 2018 it was 79, in 2019 it was 82.34 and in 2020 only 49.7; this also caused the cut for bronze medals to still be a 0 points as late as at 2 hours and 30 minutes into the contest [8].
- There was a greater incidence of contestants *forfeiting* the contest: eight contestants forfeited the contest this year, while in onsite contests this number is usually zero or one.

To the best of our knowledge, there were no cheating attempts during the contest.

Technical aspects. The server fleet was composed by a central server and a variable number of worker machines, which were used to host the *cmsWorker* service of CMS. All machines were VMs hosted on Google Cloud. The central server hosted CMS, the service for collecting the screen streams from the contestants VMs, and the service for collecting reports from *oii-proctor*. In total, we spent around 100 euro for the whole cloud infrastructure. We also set up Grafana and Prometheus for monitoring the system: you can find the dashboards at [1] and [2].

The operating system chosen for the contestants' VMs was Ubuntu MATE, due to its commonality and relatively low resource requirements. When started, it would prompt a login screen which sent the inserted credential to the central server, which would send back a Wireguard (Donenfeld, 2017) configuration file: Wireguard was used to setup a secure connection between the VM and the server and the login served to keep track of which user was on which machine. Then, during the whole duration of the contest, a service on the VM would take a screenshot every 30 seconds and send it to the central server. This allowed us to further monitor the contestants. Also on all VMs an ssh daemon was installed, to allow us to enter the machine via the Wireguard tunnel.

oii-proctor is a program written in Rust (Matsakis and Klock, 2014) and distributed as a static binary for all three major operating systems, that served as an added measure to ensure some kind of proctoring for the host machine. After asking for a login to iden-

tify the user, *oii-proctor* collects information about the running processes to check if any browser is open and if the VirtualBox process is still alive, and send this information to the central server every 15 seconds. While doing these background jobs, it also puts up a bit of a *security theater* (Schneier, 2006): to keep contestants under the impression that *oii-proctor* is constantly active doing some kind of work, thus making a would-be-cheater contestant more aware of being monitored, *oii-proctor* constantly prints meaningless information at irregular intervals of time, masking when the actual checks are done.

5. Conclusions

In this paper we described how we moved online all the phases of the Italian Olympiads in Informatics. We hope that our experience, as well as the tools developed and the other ones mentioned, can provide to be helpful to other national or local programming contest organizers.

We provide a list of online resources after the references list, including github urls of the developed tools.

References

Amaroli, N., Audrito, G., Laura, L. (2018). Fostering informatics education through teams olympiad. In: *30th International Olympiad in Informatics, IOI 2018*, 12, pp. 133–146.

Casadei, G., Fadini, B., Vita, M.G. (2007). Italian Olympiads in Informatics. *Olympiads in Informatics*, 1, 24–30.

di Luigi, W. *et al.* (2018). Learning analytics in competitive programming training systems. In: *2018 22nd International Conference Information Visualisation (IV)*. IEEE, 321–325.

di Luigi, W. *et al.* (2016). oii-web: An interactive online programming contest training system. In: *Olympiads in Informatics*, 10, 195–205.

Donenfeld, J.A. (2017). WireGuard: Next generation Kernel network tunnel. In: *NDSS*.

Maggiolo, S., Mascellani, G. (2012). Introducing CMS: A contest management system. *Olympiads in Informatics*, 6, 86–99.

Matsakis, N.D., Klock, F.S. (2014). The rust language. *ACM SIGAda Ada Letters*, 34(3), 103–104.

Prechelt, L., Phlippsen, M. (2000). JPlag: Finding plagiarisms among a set of programs. In: 2000.

Schneier, B. (2006). *Beyond fear: Thinking Sensibly about Security in an Uncertain World*. Springer Science & Business Media.

Online resources

[1] *Dashboard with CMS's metrics*. URL:
`https://snapshot.raintank.io/dashboard/snapshot/0J29w0KruEiymy6zV30beX98aRG6njiX`

[2] *Dashboard with System's metrics for CMS*. URL:
`https://snapshot.raintank.io/dashboard/snapshot/1s1wQCSYPgWdQe9f6sAeYZns5Ln1y6e5`

[3] *Dashboard with System's metrics for Terry*. URL:
`https://snapshot.raintank.io/dashboard/snapshot/kPnzTPMNAIAmOHuubdds2ltUmdJ3Q9Xc`

[4] *Dashboard with Terry's metrics*. URL:
`https://snapshot.raintank.io/dashboard/snapshot/Uw7uECvHWCbYjoNicT1DSn4ZKhdbamSd`

[5] *OII-Proctor: a portable proctoring script*. A public URL is not available for security purposes. You can ask

the source code at this email address: info@olimpiadi-informatica.it
[6] *randomTeX: a quiz randomizer for LaTeX*. URL: https://github.com/olimpiadi-informatica/randomtex
[7] *Starplag: a tool for finding the similarities between two source files*. URL: https://github.com/olimpiadi-informatica/starplag
[8] *Statistics about the Italian National Phase*. URL: https://stats.olinfo.it
[9] *Terry: a very customizable "Google Code Jam" clone useful for holding programming contests*. URL: https://github.com/algorithm-ninja/terry

G. Audrito is involved in the training of the Italian team for the IOI since 2006, and since 2013 is the team leader of the Italian team. Since 2014 he has been coordinating the scientific preparation of the OIS and of the first edition of the IIOT. He got a Ph.D. in Mathematics in the University of Turin, and currently works as a Junior Lecturer in the University of Turin.

W. Di Luigi is involved in the training of the Italian team for the IOI since 2013. He holds a Master degree in Computer Science and Engineering from Politecnico di Milano, and currently works as a Software Developer.

L. Laura is currently the president of the organizing committee of the Italian Olympiads in Informatics that he joined in 2012; previously, since 2007, he was involved in the training of the Italian team for the IOI. He is Associate Professor of Theoretical Computer Science in Uninettuno university.

E. Morassutto is involved in the training of the Italian team for the IOI since 2016, with a particular focus on the technical aspects of the contests. Since 2017 he's involved in the OIS and IIOT as scientific and technical committee member. He got a Bachelor's degree in Computer Science and Engineering at Politecnico di Milano and he's enrolled in the same Master's degree course.

D. Ostuni is involved in the training of the Italian team for the IOI since 2015, as well as focusing on the technical aspects of the contests organization. He got a Master's degree in Computer Science at the University of Milan and he's currently pursuing a Ph.D. in Computer Science at the University of Verona. He firmly believes that $\tau > \pi$.