Competitive Learning: Successes and Pitfalls from Two Years of the University of Virginia's High School Programming Contest

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Nicholas James Winschel

Spring, 2025

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman, Department of Computer Science

Competitive Learning: Successes and Pitfalls from Two Years of the University of Virginia's High School Programming Contest

CS4991 Capstone Report, 2025

Nicholas Winschel
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
pvz6tx@virginia.edu

ABSTRACT
The task of introducing Computer Science and problem-solving skills to high school students is an open-ended one with a multitude of approaches. One such approach is to run programming contests, in which students compete to solve short-form algorithmic problems. One such contest, the University of Virginia (UVA) High School Programming Contest (HSPC) is run in-person, yearly with novel problems on identically-configured machines. Organizing this event poses several technical challenges: provisioning machines, designing and implementing network architecture, generating test cases, preparing environments for problem statement development, and—most importantly—designing the problems themselves. Over the past two years of leading a team to solve these challenges, I learned a great deal about the intricacies of problem design and later applied the lessons from provisioning infrastructure to a project at my internship. Future work consists of measuring the impact of this contest on former competitors, streamlining problem development infrastructure, and continuing to run the contest in years to come.

1. INTRODUCTION

UVA's HSPC mirrors the format of the International Collegiate Programming Contest (ICPC), the oldest, largest and most well-known international programming contest (Bloomfield and Sotomayor, 2016). ICPC contests are 5-hour contests in which contestants solve problems in 3-person teams (sharing one computer) by writing code to solve 10-15 challenges. While ICPC is a contest for college students, who may have already had a formal introduction to programming and computer science, HSPC is targeted at high school students, many of whom do not have a formal computer science program at their school. At the same time, many HSPC competitors come from strong schools and may already have extensive experience competing in programming contests. From a problem-design perspective, creating a problem set that all students find engaging and informative is quite difficult.

The desired outcome of hosting an event like this is, primarily, to improve the skills of all participants, giving those early in their computer science career skills, knowledge, and a feeling of success (Kenderov, 2017). Competitions like these also give students important chances to communicate with peers and give some students "models to follow," "motivating [them] to work harder" (Kenderov, 2017). Some authors note that there is merit in the development of interesting problems themselves, with Knuth (2017) calling that task "the most critical thing."

2. RELATED WORKS

There are two in-person competitions of note: ICPC and the International Olympiad in Informatics (IOI). IOI is of a different format than ICPC, following an "Olympiad" style instead (Kenderov, 2017). High schoolers compete individually over two days to solve around six total problems. Another notable difference in format is that high schoolers

represent their country of origin, rather than (as in ICPC), their school (Kenderov, 2017; Bloomfield and Sotomayor, 2016). Our choice of problem topics is partially informed by the IOI Syllabus (Verhoef, et al., 2025).

ICPC has an extensive focus on teamwork. Different contest strategies largely differ only in roles of team members. Teamwork is critical for solving later problems and managing keyboard time effectively (Bloomfield and Sotomayor, 2016). Our choice of format is copied from that of ICPC, as we believe ICPC's emphasis on teamwork makes for a fun, collaborative experience for high schoolers.

3. PROJECT DESIGN

Effectively running the contest requires solving several technical challenges.

3.1 Challenges and Background

Generally, we wish to motivate students to solve problems, and we wish not to drag students down with unfamiliar systems or difficult-to-use infrastructure. To this end, we keep systems as standard as possible (that is, our competition systems should be like those of other high school programming competitions) and to give less-experienced competitors a chance to use the systems before the actual competition starts. We achieve this by using a well-known judging server (DOMJudge, see 3.2), a well-known competitor host image (ICPC World Finals Linux Image, see 3.3), and by running a "practice contest."

The practice contest introduces some challenges of its own. The computers are all to be reset in a short window between the practice contest and the actual contest. To do this, we add management infrastructure (see 3.3, 3.4).

The problems themselves are constructed so that as many teams as possible find the problems interesting and rewarding, but not frustrating. We address this by proposing more problems than would be necessary and cutting from the set to make it so that our whole committee finds it satisfying (see 3.6). Setting and repairing problem statements as quickly as possible requires some infrastructure (see 3.5), and this task can be unified with the task of testcase construction (see 3.7).

During the contest, only teams should be able to submit to the problems, but everyone should be able to view the scoreboard. To ensure the integrity of the contest, we firewall hosts and have them submit only via a VPN to the judging server (see 3.4). We proxy the scoreboard over a public Nginx server (see 3.4). The judging server isolates submissions from the internet, the filesystem, and each other without additional setup.

3.2 Judging Infrastructure

We use DOMJudge to judge competitor submissions and display the contest scoreboard. DOMJudge requires a "DOMServer" to manage all the contest information, and "judgehosts" to judge submissions. If judgehosts are not provisioned in sufficient number or with sufficient resources, submissions will take too long to judge, making it more difficult for teams to know if their code works. We address this by running our instance of DOMJudge in a hypervisor, with stock judgehost templates configured for quick construction if necessary. During the competition, we monitor the submission queue. If it gets too long, we initialize additional judgehosts.

3.3 Competitor Hosts

Competitor hosts are designed to maximize the chance that teams will be able to use an IDE they are familiar with. Competitor hosts are preloaded with popular IDEs such as VSCode, IntelliJ IDEA, Eclipse, PyCharm, etc. Competitors are made aware of the existence of the ICPC World Finals Contest Image ahead of the competition, so that they may practice with it if they wish. Language references are available on competitor hosts.

Competitor hosts are cleared (when necessary) via an Ansible playbook from a management host. The Ansible playbook kills the X session and clears out all directories writable by the user.

The Ansible playbook does not restart the computer.

3.4 Networking

Competitor hosts are connected to a publicly accessible virtual machine via a [formerly OpenVPN, now WireGuard for ease-of-configuration reasons] VPN. Competitor hosts are configured with host-based firewalls to only be able to communicate with the VPN host, the DOMServer and the management host.

An Nginx instance on another publicly accessible virtual machine is set to proxy the scoreboard, and only the scoreboard, from the DOMServer. This is so that even if credentials are leaked, outside individuals could not log in to submit problems for teams.

3.5 Typesetting Infrastructure

We typeset problems in LaTeX, using Kattis' problem tools to keep sample test cases consistent with statements. We use a custom script to regenerate individual problem statements/the whole problem packet when necessary.

3.6 Problemset Construction

Problemset construction is difficult. There is usually a great disparity between the team that finishes first and the team that finishes last at HSPC. We generate problem ideas from as many sources as possible: experienced competitive programmers provide interesting insights that they have seen before and interesting setups/games that give ideas for problems. We construct problems forward from these ideas. When we are lacking problems that can be solved with a specific topic, we also construct problems backward by targeting those topics.

We then sort these problems by estimated difficulty and select a subset of these to appear on the final problemset, choosing 1-2 each of "very easy" and "very hard" problems, and targeting equal proportions of "easy," "medium," and "hard" problems. Generally, we take "easy" to mean that many high schoolers with programming experience could eventually solve the problem. We take "medium" to mean that many university students with an algorithms class or two could eventually solve the problem. We take "hard" to mean that solving the problem usually requires a good amount of experience with competitive programming. We make sure that no desired topic is overrepresented or underrepresented. Finally, we change the problem statements to match the theme of the contest.

3.7 Testcase Generation

We design test cases in an adversarial fashion, developing solutions that we want to accept and solutions that we do not want to accept. We analyze the structure of the problem. Depending on the problem, it may take a specific class of test case to break certain solutions. We develop individual test case randomizers for each class of test case. We test (desired correct and desired incorrect) solutions against the testcases by using Kattis' problem tools.

We develop custom judging scripts when a given input could correspond to more than one correct output.

For problems where the user's submission interacts with an adversary (such as in a game), we develop interactors that use one or multiple strategies.

We generally seek to provide deterministic inputs; that is, if the user submits the same code (and that code is deterministic), the input to the program should always be the same. For interactive problems, the input can depend on previous outputs by the program.

4. RESULTS

We have run two competitions using this general strategy. We have scaled from <20 competitors in the first to >30 competitors in the second. In both, the judging, competitor hosts (except some physical issues), networking (except some physical issues), and typesetting infrastructure have worked flawlessly. We developed additional DHCP-based automated imaging

capabilities for the second, as the number of hosts increased significantly.

Testcase generation saw some inaccuracies, and we caution any future organizers to dedicate additional time to double-checking testcase generation. One particularly malevolent instance occurred when an organizer inserted an additional line break in a certain class of generated cases. C++ solutions (including our primary judge solution) ignored these line breaks, as Java solutions likely would have. Most Python solutions were broken by these testcases, causing issues for some teams. This was resolved quickly in contest. Custom verifiers also saw some errors. Generally, however, no incorrect solutions were accepted, which is an improvement over previous years' competitions.

Problem set construction saw significant difficulty in the first of the two competitions. We had very little data about the skills of competitors, and we had very little understanding of which problems competitors would find frustrating to code. Generally, we found that we consistently underestimated the difficulty of "easy" problems when said problems had complex implementations. For the second instance of the competition, we posited that teams would care the most about the problems that they last worked on, and we paid special attention to problems that we expected to appear at "difficulty breakpoints." In particular, we chose an interesting problem with the specific intention that 50% of teams would solve it. Our understanding of the difficulty of this problem and of the other problems was informed by the results of the previous year. This strategy proved very successful, and we found that many contestants found the problems engaging.

5. CONCLUSION

Successfully running the programming contest is vital: a smooth event is substantially more likely to engage students and promote further participation in Computer Science-related activities. Care must be taken to ensure that all components (technical and nontechnical) function in a correct and intuitive manner. A successful contest depends on working problems, problem statements, contestant systems, judging infrastructure, networking, and non-technical logistics to function. Our work over the last two years has helped future organizers understand what is necessary for each of these components. In doing so, we aim to keep the quality of future UVA HSPCs high, thereby inspiring a generation of high schoolers to pursue Computer Science.

6. FUTURE WORK

Future work consists primarily of continuing to run the contest at the same standard of quality. New effort should focus on expanding the team capacity by developing the logistics necessary to serve multiple rooms. Future instances of the competition should attempt to use some of the submission queue visualization utilities used by ICPC. Also, if printing does not return this year, next year's contest organizers should prioritize the return of printing as a contest feature.

7. ACKNOWLEDGMENTS

Thanks to Edward Lue, Richard Wang, and Miya Livingston for help with problemsetting. Thanks to Shreyas Mayya, Chase Hildebrand, Arjun Rao, Lulu Han, Ratik Mathur, Vincent Zhang, Jacob Rice, Vix Clotet, and all others who helped with systems and networking. Thanks to Param Damle, Jialin Tso, Edward Lue, and others for helping organize the nontechnical aspects of the event. Thanks to MetaCTF and the UVA Department of Computer Science for helping provide contestant computers for the event. Thanks to the UVA Computer and Network Security Club for providing hosting for central infrastructure for the event. Thanks to the UVA Department of Computer Science for helping provide funding for the event. Thanks to Rich Nguyen for acting as our Faculty Sponsor for this event. Thanks to the UVA Association for Computing Machinery and all other volunteers that provided support during the event. Thanks to all coaches and teams who attended the event and provided helpful feedback. Thanks to the creators of DOMJudge, the creators of Kattis, and the ICPC

judging staff for helping provide software used at the event.

REFERENCES

Bloomfield, A., & Sotomayor, B. (2016). A Programming contest strategy guide. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 609–614. https://doi.org/10.1145/2839509.2844632

Kenderov, P. S. (2017). Three decades of International Informatics competitions: How did IOI start. *OLYMPIADS IN INFORMATICS*, *11*(2), 3–10. https://doi.org/10.15388/ioi.2017.special.01

Knuth, D. (2017). International Olympiad in informatics: Roads to algorithmic thinking. *OLYMPIADS IN INFORMATICS*, *11*(2), 11–20. https://doi.org/10.15388/ioi.2017.special.02

Verhoef, T., Horvath, G., Diks, K., Cormack, G., Forisek, M., Lacki, J., & Peng, R. (2025). *The International Olympiad in Informatics syllabus*. International Olympiad in Informatics. https://ioinformatics.org/files/ioi-syllabus-2025.pdf