Online Submission and Evaluation System Design for Competition Operations

Zhe Chen[1], **Daniel Harabor**[1], **Ryan Hechnenberger**[1], **Nathan R. Sturtevant**[2]

[1] Department of Data Science and Artificial Intelligence, Monash University, Australia
[2] Department of Computing Science, University of Alberta, Canada

Abstract

To compare the performance of different algorithms and techniques, research communities have developed several benchmark datasets across different domains. However, tracking the progress in these research areas is not easy, as publications are shown across different venues at the same time, and many of them claim to represent the state-of-the-art. Research communities organise competitions periodically to evaluate the performance of various algorithms and techniques, thereby tracking advancements in the field. However, these competitions pose a significant operational burden. The organisers must manage and evaluate a large volume of submissions. Furthermore, participants typically develop their solutions in diverse environments, leading to compatibility issues during the evaluation of their submissions. This paper presents an online competition system that automates the submission and evaluation process of the competition. The competition system allows organisers to manage large amounts of submissions efficiently, and utilising an isolated environment to evaluate the submissions, which has been successfully used to serve several competitions and applications.

Introduction

Research in solving combinatorial optimisation problems has been a popular topic in the field of computer science and artificial intelligence. Researchers from different domains have proposed various algorithms and techniques to solve different types of problems. To compare the performance of different algorithms and techniques, research communities have developed several benchmark datasets, including the 2D Path Finding Benchmarks (Sturtevant 2012), Iron Harvest Path Finding Benchmarks (Harabor, Hechenberger, and Jahn 2022), Multi-Agent Path Finding Benchmarks (Stern et al. 2019), Voxel Benchmarks for 3D Path FInding (Nobes et al. 2023), and so on.

However, tracking the progress in these research areas is not easy, as publications are shown across different venues at the same time, and many of them claim to represent the state-of-the-art. To address this issue, research communities have organised competitions to evaluate the performance of different algorithms and techniques (Sturtevant et al. 2015; Vallati et al. 2015). These competitions often come with new benchmark problems and datasets to boost the research in the field. However, the operational burden of these competitions is high, as the competition organisers need to manage a large number of submissions, conduct performance benchmarks across all the submissions, and provide feedback to the participants. Moreover, participants often develop their solutions in different environments, which leads to all sorts of compatibility issues when evaluating their submissions.

To address these issues, this paper presents an online competition system, which automated the submission and evaluation process of the competition. The system allows participants to submit their solutions to the competition problems at any time before the due date and evaluates the submissions in an isolated environment. The instant feedback provided by the system allows participants to iterate their solutions quickly and efficiently. The system helped to bring back the Grid-based Path Planning Competition ($GPPC^2$)[1] in 2023, which was last held in 2014 and discontinued due to the high operation burden. The same architecture also serves the League of Robot Runners competition[2], which attracts more than 25 active teams worldwide with a total of 825 submissions. The competition system receives on average 9 submissions per day with a peak of 35 submissions per day during the competition period.

In this paper, we will first introduce the architecture of the competition system. Then talk about how this system is used across different applications. While running these competitions, we have faced several issues and challenges, we will discuss and share our experiences in handling them.

Related Works

There are several works in competition hosting and solver benchmarking. However, they are often designed for specific types of problems or serving specific purposes.

StartExec[3] is an open-source solving service developed at the University of Iowa, which allows benchmarking and comparing solvers on customised benchmark problems. However, the main goal of this project is to facilitate the experimental evaluation of solvers, which lack functionalities for competition management. DOMjudge[4] is an open-source programming contest jury system for International Collegiate Programming Contest (ICPC) style competitions.

[1]https://gppc.search-conference.org/
[2]https://www.leagueofrobotrunners.org/
[3]https://github.com/StarExec/StarExec
[4]https://github.com/DOMjudge/domjudge

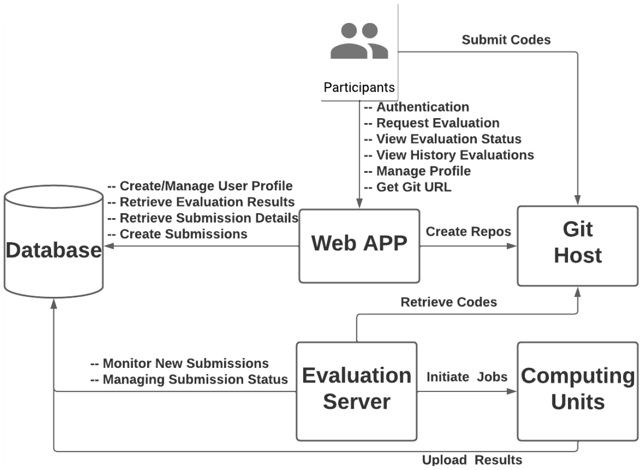

Figure 1: System Architecture

Although DOMjudge comes with a competition management system, the system is designed to evaluate simple algorithm implementations, usually a program in one file, in a restricted programming environment. The system requires competition problems documented in certain formats and judges submitted programs mainly in output correctness.

However, we aim to host competitions with solvers output complex solutions, and each submission may consist of many files and can use any dependent libraries. The performance metrics we measure are more than correctness, which includes but are not limited to solution quality, runtime, storage usage, RAM usage, preprocessing time, and so on, and different competitions we host measure submissions differently and require different evaluation processes. Modifying StartExec or DOMjudge to run our competitions is not trivial, we thus designed our competition system and aimed to provide the research community with an open-source and customisable solution.

System Architecture

The competition system is designed to be an online submission and evaluation system. The system is responsible for managing the user profiles, submission, evaluation, and results display. It aims to provide a user-friendly interface for the participants to interact with the system. The system allows participants to submit their solutions to the competition problems at any time before the due date, the submission will be queued for evaluation upon receiving it. Participants utilise a leaderboard panel to keep track of the progress of the competition, and a submission history panel to view the details of each submission.

The architecture of the competition system is shown in Figure 1. The system consists of several main components:

- **Git Host** manages git repositories for participants to submit their codes,
- **Web APP** for the users to interact with the system,
- **Database** stores participant profiles, submission data, and evaluation data,
- **Evaluation Server** monitors new submissions and initiates the evaluation jobs,
- **Computing Units** that execute the evaluation jobs in isolated Docker container and submit evaluation results.

The rest of this section will describe the details of each component.

Git Host

The Git host manages git repositories that are assigned to each participant. Participants submit codes by pushing their solutions to the git repositories. The evaluation server will then fetch the code from the git repositories upon receiving an evaluation request. In practice, it can be Github or Bitbucket, or any other git repository hosting service.

Managing submitted codes using git repositories allows efficient management of submitted codes and to track and retrieve the history of the submissions

Database

The database is responsible for storing user profiles, submission details, and evaluation data. It is also the channel components to communicate with each other. We use MongoDB[5] as the database system, which is a NoSQL database that stores data in JSON-like documents. The database contains the following main collections:

- **Users** collection stores the user profiles, including the user ID, username, email, and authentication data.
- **Competition** collection stores the competition details, including the competition ID, competition name, competition start time, competition end time, and so on.
- **Subaccounts** collection stores the subaccount details for each competition a user is involved in, including the subaccount ID, user ID, the git repository URL, competition ID, and competition-specific data.
- **Submissions** collection stores the submission details, including the submission ID, subaccount ID, competition ID, submission time, evaluation status, evaluation results, and so on.

Web APP

The web app is the main interface for the users to interact with the system. It contains two parts: a Restful API Node.js[6] back-end and a single-page web app front-end built with ReactJS[7]. The Restful API server is responsible for handling the requests from the web app, communicating with the database and communicating with the repository host, with the following functionalities:

- participant registration, authentication, authorization, and profile management,
- submission management, which creates new submission entries in the database upon receiving an evaluation request from the participants,

[5]https://www.mongodb.com/
[6]https://nodejs.org
[7]https://react.dev/

- competition management, including submission details retrieval, evaluation results retrieval, and leaderboard generation.

The single-page web app front-end provides a user-friendly interface for participants. It displays public information and competition guidelines to the users, such as the competition introduction, problem descriptions, news updates, and the leaderboard to track the progress of the competition. It also provides a user interface for the participants to register for the competition, manage their profiles, initiate evaluations, get access to their assigned git repositories, and view their submission histories and evaluation results.

To initiate a submission, participants can find the link to their assigned git repository in the web app and push their codes to the repository. Then by clicking the start evaluation button, the web app will create a new submission entry in the database and the evaluation server will be notified to initiate the evaluation process.

Evaluation Server

The evaluation server is responsible for monitoring new submissions and initiating the evaluation jobs when new submissions are found. Upon a new submission request, the evaluation server goes through the following steps:

1. Fetch the code from the git repository and record the commit hash of the code in the database.
2. Initiate an evaluation job in the database.
3. Command a computing unit to run the evaluation job.

The evaluation server also monitors and records the status of the evaluation jobs.

Computing Units

The computing units are responsible for executing the evaluation jobs, and notify the evaluation server upon the completion of the evaluation jobs. It executes a pre-defined evaluation script and submits the evaluation results to the database.

If multiple computation units are available, the evaluation server could distribute the evaluation jobs to the computing units utilising workload manager tools like Slurm (Yoo, Jette, and Grondona 2003). In this case, a shared file system across the evaluation server and computing units is required to store the evaluation jobs and results.

Docker as a Sandbox

To ensure the security of the system and construct an isolated environment for the evaluation jobs, computing units run the evaluation jobs in a Docker container. Docker (Merkel 2014) is a containerisation platform that allows us to run evaluation jobs in isolated environments.

The Docker container is built with a base image that contains the necessary dependencies for the evaluation jobs. We allow the participants to specify the names of additional dependencies in their submission and the evaluation script will install them through Ubuntu's Advanced Packaging Tool (APT).

The Docker container is also configured to limit the resources available to the evaluation jobs, such as CPU, memory, disk space, and internet access, to prevent the evaluation jobs from consuming excessive resources and prevent malicious activities from the submitted code.

To ensure a successful compilation and execution, we provide a bash script which allows participants to build the same docker container used for the evaluation jobs on their local machines. This greatly improved the efficiency of the participants in debugging runtime environment issues.

Applications

The competition system based on the proposed architecture were applied to various applications. In this section, we will introduce three systems that are based on the proposed architecture and discuss the differences across these systems.

Teaching for an AI Planning Unit

The first system is an online assignment submission and evaluation system for an AI planning unit. The AI planning unit is a university course that teaches students algorithms and techniques for solving planning and reasoning problems. The unit includes two assignments that require students to solve Multi-Agent Path Finding problems and Pacman Capture the Flag problems.

The system uses Bitbucket to manage the submitted codes and uses a simple password-based authentication system. It allows students to submit their assignment implementation at any time before the due date. It will then evaluate the submission upon receiving it and display the results on a leaderboard. The evaluation is performed on a single computer. As the accurate runtime measurement is less important here, we run multiple jobs in parallel (according to the number of available CPUs) to speed up the evaluation process. Students are allowed to make another submission after receiving the evaluation results.

The system gives immediate feedback to the students and encourages competition among the students using the leaderboard.

The Grid-based Path Planning Competition

The above system was then extended for the Grid-based Path Planning Competition(GPPC^2), an annual competition that evaluates the performance of path-planning algorithms on grid-based maps. It is a forum for tracking and disseminating state-of-the-art progress in the area. The competition measures progress in two distinct tracks:

- Classical track, which is 8-connected pathfinding on a static grid map, and
- Any-angle track, where grid paths are not restricted to eight directions.

GPPC^2 uses *Login with GitHub* as the authentication system and uses GitHub to manage the participant repositories. Participants competing in multiple tracks are assigned a git repository for each track.

Evaluation System: Different from the AI planning unit, GPPC^2 requires precise performance measurements and the runtime of Grid-based Path Planning algorithms often varies due to computational interferences. Thus the evaluation process is split into two stages on two computers: the precomputing stage and the benchmarking stage. The precomputing stage allows participants to precompute any data structure that can be used to speed up the pathfinding process. Precomputing jobs are often time-consuming and less sensitive to runtime variations, so we allow multiple precomputing jobs to run in parallel and assign even computing resources for each precomputing job. The evaluation job is then passed to the benchmarking machine and queuing for benchmark evaluation once precomputation is done. The benchmarking machine is a high-performance computer and only runs one job at a time to ensure the accuracy of the runtime measurement.

Leaderboard: The leaderboard of $GPPC^2$ is customised to compare algorithms over various metrics, such as runtime, resource usage, path quality and so on. It provides filters allowing comparisons on only undominated algorithms, optimal/suboptimal algorithms or precomputing/online algorithms.

Archives: As the system utilises git repositories to track the submitted codes and records the commit hash for each version of evaluated codes, we can easily retrieve the historical versions of the submitted code to build a post-competition code archive for future reference.

The League of Robot Runners Competition

The League of Robot Runners Competition is a competition where participants tackle one of the most complex optimisation challenges: the coordination of a large team of moving robots to fulfill tasks as efficiently as possible, while subject to computational constraints.

The competition uses the same Github-based authentication system as $GPPC^2$ and uses Github to manage the participant repositories. The major differences in the League of Robot Runners Competition are the evaluation cluster, a more informed leaderboard and the use of cloud computing.

Evaluation System: The evaluation of the League of Robot Runners Competition requires handling large amounts of simultaneous submissions and evaluating them in a timely manner. The evaluation server is connected to a cluster of computing units managed by Slurm Workload Manager (Yoo, Jette, and Grondona 2003). The evaluation server submits the evaluation jobs to the Slurm Workload Manager, which then schedules the jobs to the computing units. The computing units are virtual machines running on AWS EC2 instances, and they will execuate a predefined evaluation script to evaluate the submitted codes, submit the evaluation results to the database, and back up the raw output files to S3 cloud storage[8]. Using the AWS ParallelCluster service[9], computing units are dynamically allocated upon the demand of jobs, we allow the system scaling to 12 computing units during the peak submission period.

[8]https://aws.amazon.com/s3/

[9]https://aws.amazon.com/hpc/parallelcluster/

Leaderboard: The leaderboard used in the competition classifies results into three distinct categories, the overall best category, the line honors category, and the fast move category, and each with a different scoring function to rank participants. It also provides visualisations of the history of the competition, allowing participants to compare their performance with others over time. Additionally, we provide an all-submissions tab to allow participants to monitor who is submitting and how much progress they have made, which enhances the competition atmosphere.

Applicability to Other Competitions

The proposed competition system should apply to other competitions as long as the workflow of the competition includes submitting, evaluating, and leaderboard generation. However, the system may require modifications to adapt to the specific requirements of the competition, for example, the evaluation process, evaluation results processing, and the leaderboard generation could be different for different competitions. The differences in the evaluation output data lead to most of the changes when moving from $GPPC^2$ to the League of Robot Runners Competition. as the processing of these data is required across the system.

Issues and Challenges

When designing and running these competitions, we encountered several issues and challenges. This section will discuss some of the issues and challenges we faced and how we addressed them.

Multiple Tracks: Many competitions have multiple tracks to evaluate different aspects of the problem. In the $GPPC^2$ competition, participants can compete in both the classical track and the any-angle track. We treat each track as a separate competition entry in the Competition collection of the database. After login, participants are allowed to join and create subaccounts for each track, with each subaccount tagged with the corresponding competition ID. Each subaccount will be assigned a separate git repository for each track. Two leaderboards display the results of each track separately.

While in the League of Robot Runners Competition, we treat each round of the competition as a separate competition entry. The different categories on the leaderboard look like several tracks, but they just sort participant teams based on different criteria using the same submission data.

Periodically Competitions: Most competitions are held periodically. One can achieve this by creating a new competition entry in the Competition collection of the database or independently host the system for each competition. In the teaching unit, we simply archive and reset the database and git repositories after each semester. The $GPPC^2$ is a rolling competition without problem changes on existing tracks, thus we just create submission archives and results summaries across submissions from different years. The League of Robot Runners Competition is held annually, and we plan to create new competition entries for future years.

Export Submissions and Solutions: The competition organizers may require the export of solutions and submissions to verify the results and build a post-competition archive. Retrieving submission codes is easy as each submission records the commit hash of the evaluated code, together with the git repository that hosts the submitted codes one can easily check out each version of the submitted code. If the competition requires recording the solutions produced by the submitted code, we could simply write these solutions to the database (small output), a hard drive, or in the cloud storage, and organise them by the submission ID.

Debugging: Debugging participant submissions is another challenge, as needs to trade-off between transparency and debug feasibility. In $GPPC^2$, we provide debug instances that are similar to the evaluation instances, and each submission will be evaluated on these debug instances first and then the evaluation instances. All the output and logs on the debug instances are accessible to the participants, but all output and logs on the evaluation instances are hidden from participants. In the League of Robot Runners Competition, we provide a large amount of example instances for participants to debug their implementation offline, and hide all evaluation logs and output from the participants. Only the server operation logs on the submission are accessible to the participants. If participants cannot solve their problem with the provided debug instances or offline example instances, they could seek help from the competition organisers and the organisers could view the evaluation logs and output to help debug the problem.

Multiple Entries: In $GPPC^2$, some participants willing to submit multiple algorithms to be displayed on the leaderboard. We allow participants to create multiple subaccounts for each track (competition) and each subaccount will have a place on the leaderboard and with independent git repositories. But to prevent malicious activities from flooding the leaderboard, on default, each participant only has one subaccount for each track. They can then request permission to create more subaccounts for the same track, and a maximum number of subaccounts limitation.

Compute Limitations: Computation resources are always limited, and the evaluation process may take a long time to complete even on broken implementations. In $GPPC^2$ and the League of Robot Runners, the evaluation server will kill the evaluation job if it exceeds a certain time limit. CPU limits, Memory Limits, and Disk I/O limits are all applied through the docker container configurations.

Ranking Solutions: The competition may require ranking solutions based on multiple criteria. In $GPPC^2$, we do not rank solutions directly. Instead, we list all the metrics we collected on the leaderboard, and anyone can utilise the sorting function, and a variety of filters to rank the solutions based on their preferences. The aim is to not diminish the importance of any metric and let investigators decide which metric is more important according to the use case and the challenges they are facing. The League of Robot Runners Competition utilised multiple scoring functions to rank the solutions in different categories. In this way, we produce the winners and the competition covers a wider range of optimisation challenges.

Auditing and Rules: The competition system tries to prevent malicious activities and cheating by isolating the evaluation environment and overwriting unmodifiable files in the submitted code. However, these measures are limited, one can always find a way to cheat the system and the cost of prohibiting cheating is high. Thus we rely on the auditing process to detect cheating activities and setup rules to exclude participants who attempt to interfere with, or othewise attempt to hijack or misappropriate, any part of the evaluation system/server functionalities. By the end of the competition, we manually review the submitted codes of the top participants to ensure the fairness of the competition.

Conclusion and Future Work

Managing competitions is not easy, with many challenges to overcome and efforts to put in. In this paper, we present an online competition system that allows participants to submit their solutions to the competition problems at any time before the due date. The competition system utilises Git repositories to manage submitted codes, Web APP to interact with participants, Slurm to manage computing units, and Docker to create an isolated evaluation environment. We then discuss the issues and challenges we faced while operating and developing the competition system.

Currently, the systems used for the three applications are maintained separately, which increases the workload for maintaining these systems and the difficulty of adding new features. Thus one target of future work is to better modulate and generalise the implementation of the system so that the common component can be maintained in one place and shared among the applications. The system will then be open-sourced as a community resource to help researchers in related fields host competitions.

References

Harabor, D.; Hechenberger, R.; and Jahn, T. 2022. Benchmarks for Pathfinding Search: Iron Harvest. In Chrpa, L.; and Saetti, A., eds., *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, 218–222. AAAI Press.

Merkel, D. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239): 2.

Nobes, T. K.; Harabor, D.; Wybrow, M.; and Walsh, S. D. C. 2023. Voxel Benchmarks for 3D Pathfinding: Sandstone, Descent, and Industrial Plants. In Barták, R.; Ruml, W.; and Salzman, O., eds., *Sixteenth International Symposium on Combinatorial Search, SOCS 2023, July 14-16, 2023, Prague, Czech Republic*, 56–64. AAAI Press.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2): 144 – 148.

Sturtevant, N.; Traish, J.; Tulip, J.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The grid-based path planning competition: 2014 entries and results. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 241–250.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36(3): 90–98.

Yoo, A. B.; Jette, M. A.; and Grondona, M. 2003. SLURM: Simple Linux Utility for Resource Management. In Feitelson, D. G.; Rudolph, L.; and Schwiegelshohn, U., eds., *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, volume 2862 of *Lecture Notes in Computer Science*, 44–60. Springer.