UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO BEZERRA RIBEIRO

CONTEST ADVISOR:
A TOOL TO HELP WITH THE CREATION OF PROGRAMMING CONTESTS

CAMPINA GRANDE - PB

2021

GUSTAVO BEZERRA RIBEIRO

CONTEST ADVISOR: A TOOL TO HELP WITH THE CREATION OF PROGRAMMING CONTESTS

Trabalho de Conclusão de Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Rohit Gheyi.

CAMPINA GRANDE - PB

2021

R484c Ribeiro, Gustavo Bezerra.
Contest Advisor: a tool to help with the creation of programming contests. / Gustavo Bezerra Ribeiro. – 2021.

10 f.

Orientador: Professor Dr. Rohit Gheyi.

Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Competições de programação. 2. Contest Advisor tool. 3. Competitive programming. 4. Maratona Brasileira de Programação. 5. Brazilian Programming Marathon. 6. Codeforces. 7. Online judge plataforms. I. Ghevi, Rohit. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

GUSTAVO BEZERRA RIBEIRO

CONTEST ADVISOR: A TOOL TO HELP WITH THE CREATION OF PROGRAMMING CONTESTS

Trabalho de Conclusão de Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado em: 25 de Maio de 2021.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)

Estar preparado para competições de programação não é uma tarefa fácil, requerendo muitas horas de prática com o objetivo de melhorar as habilidades em resolução de problemas. Para ajudar nesse processo preparatório, os estudantes do ramo buscam participar de grupos de estudos, acampamentos intensivos e cursos, onde são submetidos a aulas sobre tópicos comumente presentes na resolução de problemas destas competições, além de listas de exercícios e competições não oficiais. Embora existam diversas plataformas, chamadas Juízes Online, com vastos repositórios de problemas disponíveis para prática, selecioná-los para compor uma lista de exercícios ou competição não oficial com alta qualidade geralmente é uma tarefa manual e difícil. Diante disso, o presente trabalho tem como objetivo apresentar a ferramenta *Contest Advisor* desenvolvida para automatizar tal processo de seleção de problemas e auxiliar na criação de competições de programação.

Contest Advisor: A tool to help with the creation of programming contests

Gustavo Bezerra Ribeiro
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil
gustavo.ribeiro@ccc.ufcg.edu.br

Rohit Gheyi
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil
rohit@dsc.ufcg.edu.br

ABSTRACT

Being prepared for programming competitions is not an easy task, it usually requires lots of hours practicing to improve your problem-solving skills. To help with this preparation process, students commonly join study groups, campings, or courses focused on that, where they take lessons about common programming competition topics and are submitted to lists of problems or unofficial contests. Although there are many platforms, named Online Judges, with large repositories of problems to practice, selecting them to compose your lists or contests with high quality is generally a manual and hard task. With that being said, this work aims to introduce the Contest Advisor tool that was developed to make this selection process automatic and help with the creation of such programming contests.[1]

KEYWORDS

Competitive Programming, Programming Contests, Learning

REPOSITORIES

https://github.com/gugabribeiro/contest-advisor

https://github.com/gugabribeiro/contest-advisor-service

https://github.com/gugabribeiro/codeforces-connector

1. INTRODUCTION

Competitive programming, also known as sports programming, is a mind sport where participants, commonly students from computing and related areas, are submitted to a set of well-specified problems and must be able to solve them using programming [1]. There are many official programming competitions, such as Paraiba Olympiad in Informatics [2] organized by the Federal University of Campina Grande, Brazilian Programming Marathon organized by the Brazilian Computing Society, the prestigious International Collegiate Programming Contest and the ones organized by companies such as Google Code Jam and Facebook Hacker Cup. It is important to mention that many companies evaluate the technical potential of candidates according to their problem-solving skills when submitted to problems very similar to the ones used in programming competitions [3].

Being prepared for those competitions is not an easy task and requires lots of hours practicing and studying topics that go far beyond just programming. To help with this preparation process there are many online and free platforms, called Online Judges, with large repositories of problems. Codeforces, TopCoder, and AtCoder are just a few examples of those. Their repositories usually contain problems from past official competitions or even unofficial competitions that happen on a weekly or monthly basis and helps to measure the level of each participant according to its performance on those.

However, registering on some of the aforementioned Online Judges and starting to solve problems or take part in some unofficial contests by yourself without any guidance is in the great majority of the cases, especially for beginners, not enough. And that's why students often fall back on programming competition study groups, campings and courses focused on that. All of those programs share in common the presence of coaches or experienced competitors responsible for two main things, the first one being facilitating and guiding the student through the learning process of new topics and techniques and the second consists in providing lists of problems for knowledge solidification and practicing or creating contests to simulate official competitions.

Nevertheless, create a contest for dozens of students from different levels, taking into consideration the motivational aspect, which means not to choose a set of problems hard enough to demotivate less experienced competitors and also not too easy and instead demotivating more experienced ones, also excluding all the problems they already tried or solved before and having the disposition of thousands of problems, many of them never seen before, can be a challenging task even for coaches or experienced competitors and despite the existence of many tools for own contest management such as Virtual Judge, DOM Judge, and the already mentioned Codeforces, none of then provides an easy way to do that choice.

That said, a tool to facilitate this process of selecting problems to create a contest for a set of students already known beforehand

and taking into consideration all the constraints cited previously proves to be quite useful.

2. SOLUTION

This work aims to introduce the Contest Advisor tool, which is a web platform[2] to help with programming contest creation, automatic problem set selection for a group of users already known beforehand, and follow contest standings[3]. An important thing to mention is that currently, the Codeforces is the only Online Judge integrated with the platform, meaning that all the available problems for contest creation are the ones in the Codeforces repository and also all the users who are going to participate in a contest must have an account on it, even though the Contest Advisor is already prepared to be integrated with other Online Judge platforms.

But before starting to discuss the proposed tool in more detail and give a brief description about its features, it becomes necessary to understand why Codeforces was chosen to be the pilot Online Judge of the solution.

2.1 Codeforces

Codeforces is one of the most popular online judge platforms nowadays and it is very hard to find a programming competitor that does not have an account on it. The popularity of the platform is given by hosting, almost every week, unofficial contests and using the performance of its contestants on those to measure their levels. To do that, Codeforces uses a Rating System very similar to the Elo Rating System used in chess and developed by Arpad Elo [4]. In a few words, a contestant is characterized by its rating R and given two contestants A and B with their respective ratings being R_A and R_B the probability that contestant A will get a higher position in an unofficial contest than the competitor B is given by:

$$P_{AB} = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

That said, it turns out to be possible to calculate the expected position of each contestant in a contest and also the difference between its real and expected positions. Depending on this difference, the contestant's rating will increase or decrease, the bigger this difference in absolute values, the bigger the change [5].

Another important characteristic of Codeforces is that it provides a repository of more than 6,900 problems from past unofficial contests. All of them are tagged with related topics, meaning that it is possible to solve the problem using a technique or algorithm related to that topic. And the most important thing, the problems also have a rating or difficulty level on the same scale as the contestant's ratings, calculated according to the ratings of the contestants who solved that problem during an unofficial contest.

[2] The Contest Advisor web platform was deployed on Heroku [6] and can be accessed on: https://contest-advisor.herokuapp.com/

[3] A screencast with a demonstration of the Contest Advisor's features can be accessed on: https://youtu.be/36UtM2910gk

Those characteristics, combined with the fact that the platform also provides an API (Application Programming Interface) to retrieve information about their contestants and problems, made the choice of Codeforces to be the pilot Online Judge for the Contest Advisor tool very easy.

2.2 Features

Contest Advisor has two types of users, Authenticated and Anonymous. To be an Authenticated user you do not need to create an account into the platform, providing email, password, and other information, the only thing that is required is an already existing Google account.

Authenticated users can list contests they previously created in the platform and also create new ones. Every contest listed contains information like their name, start time and duration, the contestants who participated or are going to participate, its problem set, and a link for its standings.

There are two ways that Authenticated users can create contests, the first one consists in providing in advance the handles (Codeforces contestant identifier) of all the contestants who are going to participate, the problem topics to consider during the choice, and the number of problems and let the tool automatically select a set of problems according to the provided data. The second way to create a contest is manually providing the problem set, with no restriction on disposing of in advance the contestants who are going to participate. An Authenticated user can still add new problems or contestants to a contest already created, but note that in the case where the problem set was selected automatically by the Contest Advisor there will be no guarantee that the new set of problems.

Anonymous users are allowed to see any contest standings, meaning that the participants of any contest are not required to authenticate on Contest Advisor, they will be able to see the contest standings and access its problem set without any problem.

3. AUTOMATIC PROBLEM SET SELECTION

This section aims to provide an explanation of how the automatic problem set selection works. To do that we define the selection criteria to guarantee a good selection of problems and briefly explain the selection strategy adopted according to those criteria.

3.1 Selection Criteria

A set of competitors, a set of topics to consider, and the number of problems to select defines some important selection criteria.

Given a set of all the available problems to the choice and according to the provided set of competitors in the attempt to consider only problems that have never been seen by any of them, should be disregarded from the available problems all the problems that were solved or tried by at least one of the competitors. The problems whose topics do not include any of the provided topics should also be disregarded. From the remaining set of available problems, the number of problems provided should be selected.

Considering that the level of the competitors provided may differ between them and extolling the importance of the motivational aspect of the choice, the selected set should follow other important criteria. It should not be too hard because it may demotivate the less experienced competitors or the opposite being too easy, and instead, have a high chance of demotivating the most experienced ones. In other words, should be considered only balanced sets of problems.

To measure how balanced a problem set is, the following metric was created, considering *C* the levels of a set of competitors and *P* the levels of an arbitrary subset of the available problems following the same scale of the competitor's levels:

$$ProblemSetScore = \sum_{i=1}^{|C|} \sum_{j=1}^{|P|} (C[i] - P[j])$$

A positive value of this metric means that the problem set is easy for that set of competitors. On the opposite, a negative value means that the set of problems is hard for those competitors. Finally, a *ProblemSetBalance* equals 0 means that this problem set meets our balance criteria.

3.2 Selection Strategy

Notice that our original selection problem has been reduced to an optimization problem where we need to find a set of problems that satisfies the selection criteria adopted.

For the sake of simplicity, let's suppose that the set of available problems that we are going to consider has already been passed through some process to guarantee the first two criteria mentioned earlier (non solved or tried problems by any of the provided contestants nor problems whose topics do not match any of the provided topics). That said, it remains the task of finding the best set of *N* problems, in terms of the minimum absolute value of *ProblemSetBalance*, among the set of *S* available problems.

Given that, the simplest strategy to solve this problem is the Complete Search or Brute Force [7] approach where all the combinations of *N* problems from the set of the available problems are analyzed and the best one according to our criteria is chosen. But it turns out that this solution is not feasible for our constraints because of the binomial nature of the number of combinations.

Another strategy can be derived from the fact that the summation of the differences between a set of values and their arithmetic mean is always zero. This means that if we select *N* problems all of them with the same level and that level being equal to the arithmetic mean of the contestant's levels the *ProblemSetBalance* of the selected set will always be zero. But, no matter how, since all the problems have the same level in this set we are going to have a monotonous set which in general is not good for a contest

However, this result is important to us because it shows that if we select problems with equidistant levels from the mean of the contestant's levels the *ProblemSetBalance* will also tend to 0. That said, the problem is now reduced to find a set of problems with their levels equidistant to the mean of the contestant's levels, which allows us to solve this by statistic approximation using a Normal or Gaussian Distribution:

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

To define a Normal Distribution we need two values: mean or mathematical expectation (***μ***) and standard deviation (***σ***) [8]. Since we want the problem's levels to be equidistant from the contestant's levels we can use the contestant's levels to be the mean of the distribution.

Considering the characteristics of the Normal distribution and its curve:

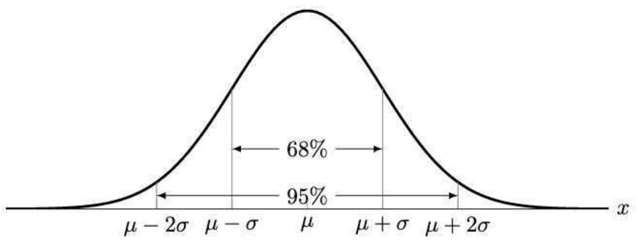


Figure 1. Normal Distribution Curve

If we use the difference between the level of the most experienced contestant and the mean of the contestant's levels as the standard deviation of the distribution, we are going to achieve a set with expected 68% of the problems being in the range of difficulty from ***μ*** - ***σ*** to ***μ*** + ***σ*** inclusive, which means that is expected that 16% of the problems will be harder enough to challenge the most experienced contestants.

That said, our final strategy consists of getting randomized samples of *N* levels following a Normal Distribution with mean equals to the mean of the contestant's levels and standard deviation equals to the difference between the level of the most experienced contestant and the mean of the contestant's levels between all the samples, selecting the one with the minimum absolute value of *ProblemSetBalance* and finally for each level in the chosen sample select any problem with that level.

4. ARCHITECTURE

The architecture adopted to develop the Contest Advisor follows a Client-Server model, where the client requests one or more resources to a specific service, and a server responsible for it handles the request and responds with the requested resources [9]. The communication between the two components is made following the HTTP (HyperText Transfer Protocol) using the REST (Representational State Transfer) model.

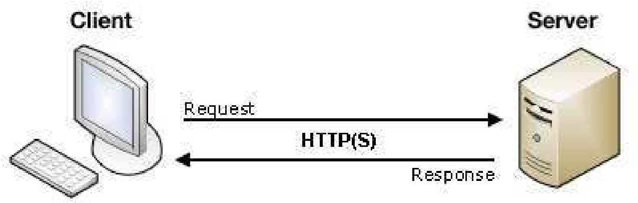


Figure 2. Client-server model using HTTP(S)

There are two main components in the architectural project, the first one is the frontend that assumes the client role in the arrangement and is responsible to present. in a friendly way, the resources requested as the result of the user's interactions on it. The second is the backend that assumes the server role of the architecture and is responsible to handle all the client's requests using a set of pre-developed business rules behind an API abstraction, the Storage, and the Online Judge Gateway.

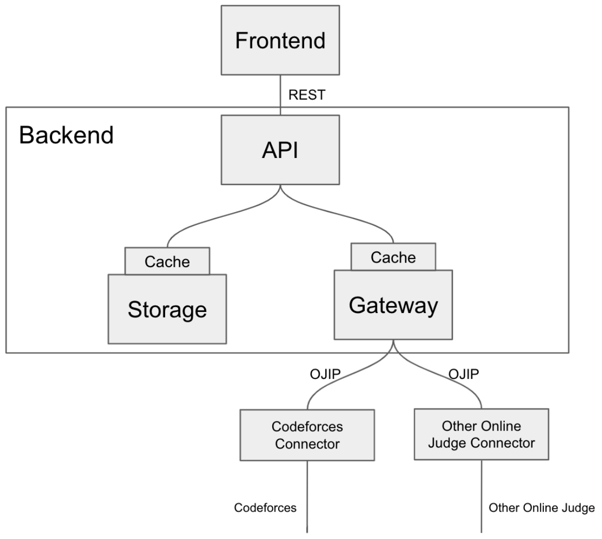


Figure 3. General representation of Contest Advisor architecture

4.1 Frontend

The frontend was developed using ReactJS, which is an open-source JavaScript library maintained by Facebook to build user interfaces [10]. This technology is based on components and this provides a flexible and modular development with a high capacity for reuse and with a context API very easy to use and perfect to maintain the application state without the need for other state management libraries like Redux.

Tachyons, which is a Cascading Style Sheets (CSS) [11] design system used to stylize components, was adopted to structure the user interface and prevented from writing a single CSS line during the entire development alongside the React-Bootstrap [12] component library, responsible for creating pure React components using the Bootstrap CSS framework [13] without the need of any unnecessary dependencies.

To perform HTTP requests to the server-side, the native JavaScript Fetch API was used.

4.2 Backend

The backend was developed using vanilla JavaScript on top of NodeJS that is an open-source JavaScript runtime [14]. Those technologies were selected for their easy configuration, usage and also for being widely used in the community, facilitating the search for solutions in forums and platforms such as Stack Overflow.

4.2.1 REST API

The REST API was developed using Express, which is an open-source framework for NodeJS designed to build web APIs. It provides a very straightforward way to define with a few lines of code the router and their respective handlers or controllers responsible to handle the API requests, in this case, following the REST model.

4.2.2 Cache

The Cache layer on top of both Storage and Gateway subcomponents, to improve the performance of the system, was implemented using Redis, which is an open-source in-memory data structure store, used as a distributed in-memory key-value database, cache, and message broker. The idea here is caching some specific costly responses, in terms of latency and/or processing, from the Storage and Gateway subcomponents. The client used to connect the backend application with Redis was the IORedis open-source client for NodeJS.

4.2.3 Storage

The Storage subcomponent, responsible to persist all the application data, was implemented using PostgreSQL, which is an open-source relational database. The choice of PostgreSQL [15] was based on the fact that it handles the data persistence using a transactional model aiming to guarantee data consistency and by the well-defined and well-structured nature of the entities and their respective relationships in the system. The Sequelize ORM [16] (Object-Relational Mapping) was used to provide a database abstraction to the service [.

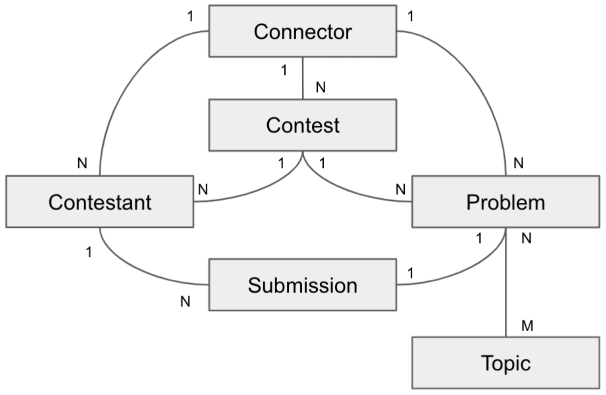


Figure 3. Contest Advisor entities model

4.2.4 Gateway

The Gateway subcomponent is an internal abstraction of the system that is responsible to communicate with the external entities outside the architectural arrangement, those entities being the Online Judges.

However, since each Online Judge has a different data format and its way to provide this data it becomes necessary to define a convention capable of facilitating the integration of them with the Gateway subcomponent. This convention occurs through two definitions, the Online Judge Integration Protocol and the Online Judge Connector, both proposed and developed in this work.

4.2.4.1 Online Judge Integration Protocol

The Online Judge Integration Protocol (OJIP) defines a set of REST Resources and its formats in a way that the Gateway subcomponent is capable of handling, being them:

- **Problems**, where a GET request on the */problems* resource should be able to retrieve all the problems from the repository of an Online Judge as a JSON array of objects with:
    - *id* (id of the problem);
    - *name* (name of the problem);
    - *level* (level of the problem as a numeric value in the same scale of the contestant's level);
    - *topics* (name of the topics as an array of strings).
- **Contestant Profile**, where a GET request on the */contestant/:id* resource should be able to retrieve the profile of the contestant identified by its *id* on an Online Judge as a JSON object with:
    - *name* (name of the contestant);
    - *level* (level of the contestant as a numeric value).
- **Contestant Submissions**, where a GET request on the /contestant/:id/submissions resource should be able to retrieve all the submissions from the contestant identified by its *id* on an Online Judge as a JSON array of objects with:
    - *problemId* (id of the problems associated with that submission);
    - *momentInSeconds* (the moment where the submission was made as an Epoch);
    - *verdict* (the verdict of the submission, being SOLVED if that submission solved the problem or TRIED, otherwise).

4.2.4.2 Online Judge Connector

The Online Judge Connector is an abstraction responsible to implement the OJIP specifications and request the original resources from an Online Judge according to its specifications.

In a few words, it should provide a REST API with the resources defined in the aforementioned integration protocol to the Gateway subcomponent.

After its implementation, to finish the integration process, the Online Judge Connector should be registered using the Contest Advisor API to persist its metadata:

- *name* (Name and unique identifier of the connector through the system);
- *URL* (The accessible location through the internet where the connector is hosted).

Done that, the Online Judge behind the connector is considered integrated with the platform and should be possible to create contests using its repository of problems or even automatically select problems according to a set of contestants from the Online Judge itself.

5.1 Authentication

The Firebase Authentication [17] platform was used to provide authenticated access for the Contest Advisor users. The decision to outsource the authentication to the Firebase tool was made based on the simplicity of its integration and usage since it provides an easy way to use your Google account to create access credentials that can be used throughout your application.

In a few words, the frontend application pops up a built-in Firebase Authentication UI for the user to select what Google account they want to use, the user will input the Google Account's password if needed and after that, it will be logged in into the platform as an Authenticated user.

On the backend side, every authenticated request will receive an authorization credential token as a request header from the client-side and the backend itself will call the Firebase Authentication service to check if that token credential is valid for that request or not.

5. EVALUATION

This section aims to introduce the production environment, the methodology used to evaluate the quality of a problem set selected automatically and the results obtained during a real usage of the Contest Advisor according to this methodology.

5.1 Production Environment

All the products developed in this work were deployed in production using Heroku, which is a cloud Platform as a Service (PaaS) that supports multiple programming languages and enables developers to build, run and operate applications entirely in the cloud [6]. It provides a limited, but free version of its services that was used to deploy the frontend application, the backend service application alongside its database, and the Codeforces connector. The only component of the system that was not deployed in Heroku was the Cache layer, which was deployed in a limited and free environment of Redis Labs, which is a private software company that provides infrastructure management for Redis environments.

5.2 Methodology

To evaluate the quality of a problem set selected automatically by the Contest Advisor tool, a subset of 9 students from the Olympic Project in Informatics and the Advanced Algorithms class of the 2020.3 academic period of the Computer Science Course, both from the Federal University of Campina Grande was selected.

The students from the selected subset were submitted to a 5-hour contest with 12 problems automatically selected by the tool according to the defined criteria. It is important to mention that the student's levels varied from 1,600 to 2,400 Codeforces rating points.

After the contest, the students were asked to evaluate their satisfaction with the quality of the selected problem set according to a set of well-defined aspects:

- ***Challenginess***: the capacity of a set of problems from a contest to provide a challenge to its contestants;
- ***Multidisciplinarity***: the capacity of a set of problems from a contest to require solutions using multiple approaches, techniques, and algorithms;
- ***Variability***: the capacity of a set of problems from a contest to be from different levels taking into consideration the contestant's levels.

There were 5 possible qualitative answers for each of the aforementioned aspects and the students were asked to choose only one for each aspect:

- Very unsatisfied;

- Unsatisfied;
- Neutral;
- Satisfied;
- Very Satisfied;

5.3 Results

The results obtained show that in general, the students showed high satisfaction taking into consideration the defined aspects:

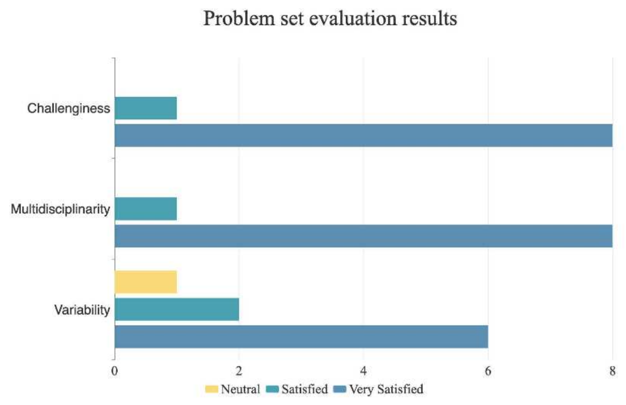


Figure 4. Chart with the problem set evaluation results

Grouping the students by their ratings in the following way:

- **Less Experienced**: from 1,600 to 1,900 rating points;
- **Experienced**: from 1,901 to 2,200 rating points;
- **More Experienced**: from 2,201 to 2,400 rating points.

And observing only the *Variability* aspect, there will be a tendency of having better results in terms of satisfiability from the *Less Experienced* students to the *More experienced* ones:

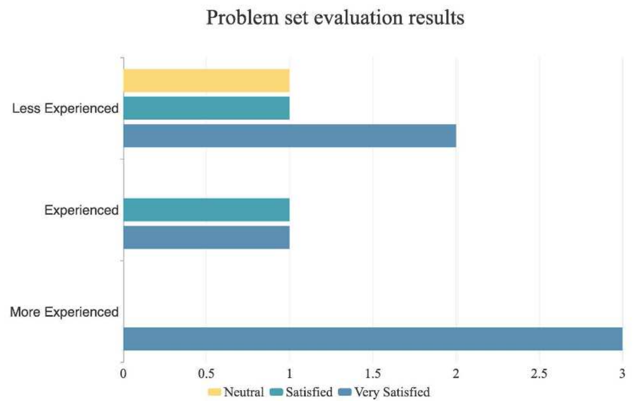
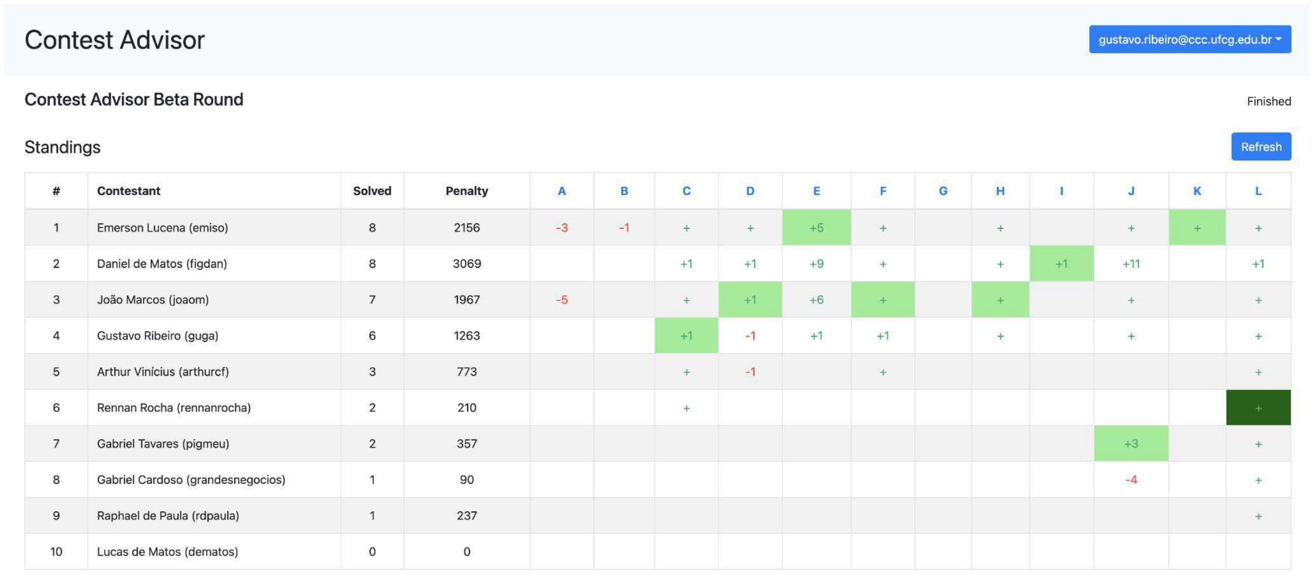


Figure 5. Chart with the problem set evaluation results for Variability

This may indicate that the problem set was better selected for a set of students with higher experiences.

But note that this does not mean that the less experienced ones were unsatisfied with the selection, but instead that they are less satisfied than the other students and this is an indication that other approaches to build the Normal Distribution responsible to select the problem's levels using statistical approximation should be considered to achieve better results.



Contest Advisor Beta Round

Finished

Standings

#	Contestant	Solved	Penalty	A	B	C	D	E	F	G	H	I	J	K	L
1	Emerson Lucena (emiso)	8	2156	-3	-1	+	+	+5	+		+		+	+	+
2	Daniel de Matos (figdan)	8	3069			+1	+1	+9	+		+	+1	+11		+1
3	João Marcos (joaom)	7	1967	-5		+	+1	+6	+		+		+		+
4	Gustavo Ribeiro (guga)	6	1263			+1	-1	+1	+1		+		+		+
5	Arthur Vinícius (arthurcf)	3	773			+	-1		+						+
6	Rennan Rocha (rennanrocha)	2	210			+									+
7	Gabriel Tavares (pigmeu)	2	357										+3		+
8	Gabriel Cardoso (grandesnegocios)	1	90										-4		+
9	Raphael de Paula (rdpaula)	1	237												+
10	Lucas de Matos (dematos)	0	0												

Figure 6. Standings of a contest held on Contest Advisor used to evaluate the quality of an automatically selected problem set

6. EXPERIENCE

This section aims to describe the experience of developing the Contest Advisor application with a focus on the development process, the main challenges and limitations of the work and the future works planned.

6.1 Development Process

For the implementation of the system, an agile methodology was adopted using the principles of Scrum [18]. This methodology is based on fast deliveries of small features, which are evolving incrementally and iteratively until reaching the final product.

The idea behind Scrum consists of following development cycles called Sprints, with short and well-defined intervals, generally weekly, biweekly, or monthly where the tasks are executed.

All the features that were developed during the project, as their priorities, were defined in the Product Backlog during the phase of elicitation of requirements of the tool. The Sprints were defined to have cycles of the size of a week and a subset of the tasks from the Product Backlog was selected to compose each Sprint Backlog.

The approach used to select the tasks for each Sprint Backlog was a bottom-up approach where the backend tasks were selected first to complete the development of the backend components and after that, start the development of the frontend application.

6.2 Main Challenges and Limitations

The design of an architecture capable of providing easy integration with other online judge platforms and developing the application using technologies never used before, like Redis, PostgreSQL and Heroku were the main challenges of this work.

Creating contests with problems from different online judges, for example, is one of the main limitations currently. Although it is already possible to integrate with other online judges using the OJIP protocol, it is not possible to create contests using problems from different online judges, nor automatically recommend problems.

The performance of displaying the contest standings is also another limitation. Currently, a cache policy of 60 seconds of expiration time is being used to prevent multiple unnecessary calls to the online judge's connectors and be throttled by the online judge itself. However, this implies that all contest's standings will be updated only after 60 seconds periodically, and this may affect the results of the contest a little bit since the contestants will not be able to see what problems were solved so far until the cache expires.

The creation of new contests and the automatic problem set selection features misses some important validations. One of them is if the contestants provided exist, and this may negatively impact the recommendation algorithm since one of the parameters used is the mean of the contestant's levels.

The platform used to deploy the application also has some limitations, mainly for being a free version. The size of the database and cache store, the load the system can handle and the latency of the services, since the system can enter in an idle state, are among the main limitations imposed by the platform.

6.3 Future Works

The current scope of Contest Advisor was developed simplistically. However, the system has the potential to be used in many use-cases, and therefore improvements in the user interface can be adopted in future work. In addition, new improvements can be incorporated, such as:

- Integrate other online judges with the platform;
- Allow automatic recommendation of problems and contest creation with problems from different online judges;
- Parameterizing the recommendation strategy;
- Improve standings performance.

ACKNOWLEDGMENTS

We thank Dr. Rohit Gheyi for the invaluable guidance. We thank our institution, the Federal University of Campina Grande, and its Computer Science undergraduate course, for the support. We thank our colleagues from the Advanced Algorithms classand from the Olympic Project in Informatics for helping during the evaluation process of this work.

REFERENCES

[1] HALIM, Steven; HALIM, Felix. Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s, 2018.

[2] Paraiba Olympiad in Informatics, 2008-2021. http://www.dsc.ufcg.edu.br/~opi/. Last access: May 17th, 2021.

[3] MCDOWELL, Gayle Laakmann. Cracking The Coding Interview: 189 Programming Questions and Solutions. Palo Alto, CA: CareerCup, LLC, 6th Edition, 2016.

[4] ELO, Arpad. The Rating of Chessplayers, Past and Present. New York, NY: Arco. 2nd Edition, 1986.

[5] MIRZAYANOV, Mike. Codeforces Rating System. https://codeforces.com/blog/entry/102. Last access: May 12th, 2021.

[6] Heroku, 2007-2021. https://www.heroku.com/. Last access: May 17th, 2021.

[7] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to Algorithms. Cambridge, MA: MIT Press, 3rd Edition, 2009.

[8] BRYC, Wlodzimierz. The Normal Distribution: Characterizations with Applications. New York, NY: Springer, 1995.

[9] PUDER, Arno; RÖMER, Kay; PILHOFER, Frank. Distributed Systems Architecture. Burlington, MA: Morgan Kaufmann Publishers, 2005.

[10] Facebook Inc, 2017-2021. https://reactjs.org/. Last access: May 17th, 2021.

[11] W3C: Cascading Style Sheets, 1994-2021. https://www.w3.org/Style/CSS/ Last access: May 17th, 2021.

[12] React-Bootstrap. https://react-bootstrap.github.io/. Last access: May 17th, 2021.

[13] Bootstrap. https://getbootstrap.com/. Last access: May 17th, 2021.

[14] OpenJS Foundation, 2019-2021. https://nodejs.org/en/ Last access: May 17th, 2021.

[15] The PostgreSQL Global Development Group. 1996-2021. PostgreSQL. https://www.postgresql.org/. Last access: May 17th, 2021.

[16] Sequelize ORM, https://sequelize.org/. Last access: May 16th, 2021.

[17] Firebase, 2010-2021. https://firebase.google.com/. Last access: May 17th, 2021.

[18] SUTHERLAND, J. J.; SUTHERLAND, Jeff,. Scrum: The Art of Doing Twice the Work in Half the Time, 2014.