



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

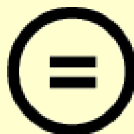
다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#) 

Master's Thesis

LEARNING HOW STUDENTS ARE LEARNING IN PROGRAMMING LAB SESSIONS

SeyedMoein Mirhosseini

Department of Computer Science and Engineering

Graduate School of UNIST

2018

LEARNING HOW STUDENTS ARE LEARNING IN PROGRAMMING LAB SESSIONS

SeyedMoein Mirhosseini

Department of Computer Science and Engineering

Graduate School of UNIST

Learning How Students Are Learning in Programming Lab Sessions

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

SeyedMoein Mirhosseini

12. 15. 2017

Approved by



Advisor

Sam H. Noh

Learning How Students Are Learning in Programming Lab Sessions

SeyedMoein Mirhosseini

This certifies that the thesis/dissertation of SeyedMoein Mirhosseini
is approved.

12. 15. 2017

signature

Advisor: Sam H. Noh

signature

Sungahn Ko: Thesis Committee Member #2

signature

Jiwon Seo: Thesis Committee Member #3

signature

Abstract

Programming lab sessions help students learn to program in a practical way. Although these sessions are typically valuable to students, it is not uncommon for some participants to fall behind throughout the sessions and leave without fully grasping the concepts covered during the session. In my thesis, I will be presenting LabEX, a system for instructors to understand students' progress and learning experience during programming lab sessions. LabEX utilizes statistical techniques that help distinguishing struggling students and understand their degree of struggle. LabEX also helps instructors to provide in-situ feedback to students with its real-time code review. LabEX was evaluated in an entry-level programming course taken by more than two hundred students in UNIST, establishing that it increases the quality of programming lab sessions.

Contents

Abstract	i
List of Figures	iv
List of Tables.....	v
List of Algorithms	vi
I. Introduction.....	1
II. Background and Related Works	3
III. Design Goals	4
IV. Understanding Struggling Students	5
V. Estimating Distribution of Problem Solving Times.....	5
5.1 Estimating Parameters with Order Statistics.....	7
5.2 Modeling Students Struggle	8
VI. Qualitative Code Review	9
6.1 Clustering Students' Solution.....	9
6.2 Comparing Code Clustering Algorithms	11
6.3 Reviewing with Clusters.....	12
VII. LabEX User Interface	12
7.1 Students Interface	13
7.2 Lab Staff Interface	13
7.3 Mobile View	14
VIII. Evaluation	14
8.1 Case Study.....	15
8.2 Real-World Experiment.....	16
8.3 Methodology	16
8.4 Qualitative Observations	17
8.5 Post-Study Interview	18
8.6 Discussions and Limitations	19
IX. Conclusion.....	19
Acknowledgement	20
References.....	21

List of Figures

Figure 1: A visualization of time-window analysis of TA's time in two sessions -----	1
Figure 2: Assessment of Modeling Problem Solving Times with A Log-Normal Distribution -----	6
Figure 3: Snapshot of Code Reviewing Interface -----	12
Figure 4: Snapshot of Progress Overview -----	13

List of Tables

Table 1: Students' Task Completion Times in a Session (Min.)-----	5
Table 2: Evaluation of Clustering Algorithms. -----	11

List of Algorithms

Algorithm 1: LabEX's Code Clustering Algorithm	10
------------------------------------------------------	----

I. Introduction

In recent times, higher education institutions hold programming lab sessions to help students learn to code in a first-hand manner. Students learn as much as, if not more, in these sessions than by listening to lectures [1]. This further emphasizes the significance of practice in the context of computer science education. In a typical lab session, participants are provided with a number of coding tasks and instructors and teaching assistants (TAs) help them throughout the session when necessary.

In a typical lab session, an assistant helps ten or more students, therefore, it is difficult to always reach out to all participants who need support. This becomes more evident in case of large-scale lab sessions where students have diverse coding and problem-solving skills. Additionally, they possess different cultural and personal thresholds for asking questions. As a result, students have a very different course of solving the tasks - some may be stuck on an introductory task while others have finished all the problems. Participants ask questions at a very different rate as well - some ask simple syntactic questions very often while others may not ask any question even if they are struggling very much.

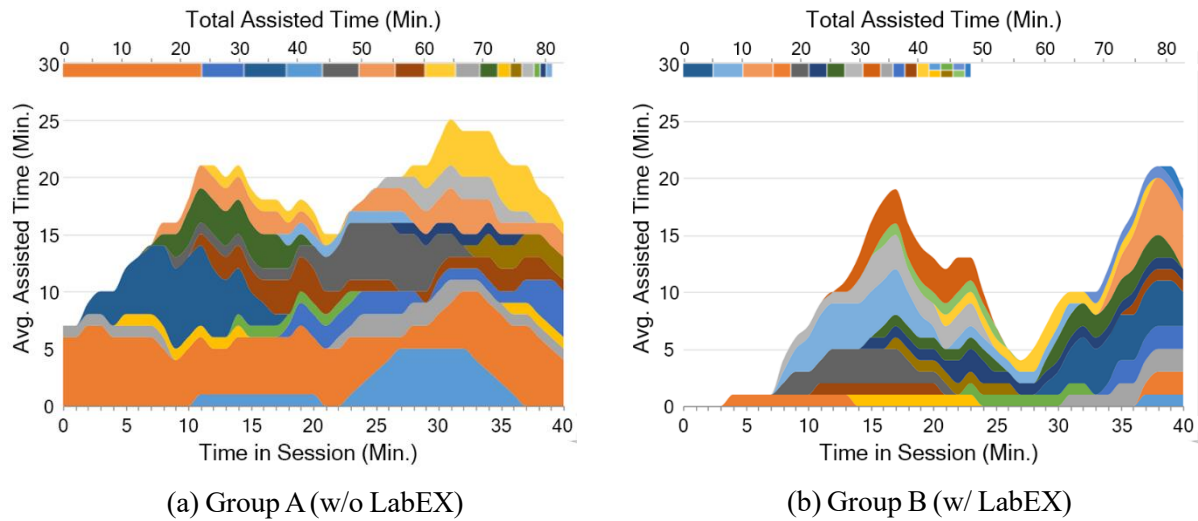


Figure 1: A visualization of time-window analysis of TA's time in two sessions. A stacked area shows the amount of time TAs helped each student in 10-minute window over time.

Figure 1a shows the amount of time TAs spent to help each student in our experimental lab sessions. Clearly, a large amount of TAs' time is monopolized by a single student (orange color); the student asked a lot of simple questions, many of which he could have figured out by himself. We observed that other students in the lab were struggling and needed help, but TAs were not aware of it at the moment because 1) those students did not ask for help, and 2) some TAs were occupied with helping the monopolizing student.

It is generally a non-trivial task for lab assistants to have a comprehensive picture of overall students' progress. An instructor can easily have a wrong impression of how students are doing based on his/her limited, biased interaction with students – students asking a lot of basic questions or those leaving the session early after finishing all the tasks. More importantly, it is even more difficult for lab instructors to qualitatively assess and help improve students' solutions during the lab session since feedback are usually provided to only those participants who ask for help. The works of silent students who never ask for help/feedback are unlikely to be reviewed by a lab staff.

This dissertation presents LabEX, a lab-session monitoring system that helps instructors better understand how students are learning in real-time by providing them with a comprehensive view of the lab sessions. With LabEX, instructors can easily inspect how students are progressing with their tasks; LabEX also helps with identifying struggling students and notifying lab staff so that they can give assist them at once. In addition, LabEX summarizes students' solutions for instructors enabling them to readily distinguish poor ones and provide students with timely feedback. The contributions of this thesis are the design, implementation, and evaluation of LabEX. The specific list of contributions is as follows:

Modeling students struggle. LabEX employs statistical methods to help spotting struggling students as well as their degree of struggle. Using the order statistics [2] of the completion times for each task, LabEX estimates their distribution. That helps to identify students who have fallen behind their peers wiring on a task. LabEX uses more detailed information about the students' interactions and time spent on the task to rank the struggling students.

In-situ code review and feedback. A code clustering technique is used by LabEX to group similar solutions and display them to lab assistants in order to help them with qualitative review of students' solutions. Clustering similar solutions, helps assistants to readily identify creative solutions and share them with participants. They can also detect poor solutions and advise the students to improve their work. We believe that such in-situ feedback greatly improves participants learning experience in the lab.

Large-scale real-world experience. LabEX was tested in an introductory-level programming course taken by 272 freshmen in one semester at our institute. LabEX was successfully employed in the programming lab sessions for the course (36 sessions in total) helping 17 lab staff (16 TAs and one instructor). The study reveals that LabEX helps to significantly improve the lab session experiences for both lab assistants and students. A separate controlled experiment was also conducted with a goal to assess LabEX in a quantitative fashion. The experiment results confirm that LabEX assists lab staff to efficiently use their time to help students.

II. Background and Related Works

There has been vast amount of studies on improving learning experiences in the context of large-scale lab classes, computer science education, and learning in MOOC (Massive Open Online Courses). Here is a summary of the most relevant work and description of the background of this study.

In several domains, situational awareness tools help users comprehend and respond to their surrounding situation [3]. These tools are used especially in mission critical domains, such as monitoring systems for power plant operators and air traffic controllers. In educational contexts, similar tools are designed to assist instructors to better understand their audience and teach more efficiently. Codeopticon [4] increases a tutor's situational awareness in MOOC teaching sessions while watching hundreds of learners writing code in real-time. LabEX aims to improve lab staff's situational awareness for offline lab sessions having different teaching context.

With the rise of MOOC, there have been a lot of studies on improving the quality of learning experience in MOOCs. Generally, many researchers strive to classify different groups of learners by inspecting students' engagement and behaviors [5] [6] [7]. A particular challenge in MOOC is high dropout rate of learners. DropoutSeer [8] employs visual analytics methods to help instructors better understand learning patterns related to learners' dropout behavior.

Receiving appropriate feedback is an important part of learning experience. In classrooms where the number of learners significantly exceeds the number of instructors and TAs, manual assessment of students' work and providing them with qualitative feedback is quite demanding, especially in programming courses where students may come up with many different solutions. In [9] [10], clustering techniques are used to help instructors explore the variations among thousands of solutions submitted in MOOC, enabling them to provide appropriate feedback. Although the techniques applied in our work are inspired from prior work, LabEX aims to improve offline lab sessions with real-time feedback. More specifically, we provide instructors with a summary of different solutions in real-time during a lab session so that they can provide students with timely and personalized feedback.

Jupyter Notebook [11] is an interactive interface that allows users to write and execute codes online. Its usefulness has made many to utilize it in different domains such as machine learning and statistical modeling. Jupyterhub [12] builds a multi-user hub which creates a single Jupyter notebook server for everyone in a group. Students of a programming course for instance, can connect to Jupyterhub and write their solutions in a Jupyter notebook.

LabEX effectively employs Jupyterhub as one of its main building blocks where each student is provided with a Jupyter notebook server upon logging in. Jupyter's interface provides students with more convenience to write, run and submit their solutions as opposed to the UNIX command line environment which is usually used in typical programming lab sessions.

III. Design Goals

Before LabEX was designed, we studied the programming lab sessions by participating in them and observing how they are run as well as interviewing few lab assistants. We attended five lab sessions of an introductory-level programming course in our institute and observed the whole sessions. The sessions started with a summary of each week's lessons delivered during the lectures, followed by an hour and a half of problem-solving by the participants. The major observations are listed as below:

- The instructor and TAs were busy answering questions and clarifying tasks during the entire sessions.
- 20% of the students asked more than 80% of the questions; the questions by the 20% students are often very trivial, such as questions on simple syntax errors, implying that they ask questions too readily instead of spending some time to think over them by themselves.
- Students were overlooked a few times when they raised hand to ask questions because all assistants were busy.
- Some students seemed to be stuck at a problem and were reluctant to ask for assistance; as a result, they left without having their tasks completed.

To conclude, the instructor and TAs could not have a comprehensive view of lab sessions and spent much time assisting a small number of participants who asked questions. They did not have the time, for instance, to share their opinion about students' progress or difficulty of tasks with other lab staff.

An interview with the lab assistants at the end of the semester confirmed our findings. One TA commented, "Sometimes students are stuck in a difficult problem for long, and we only noticed it at the end of the session." Another TA said, "The progress among students differs very much, and I want to be able to find and help those who are behind." The TAs and instructor both mentioned that they are too busy during lab sessions and cannot afford to discuss their thoughts on, for example, what aspects of the tasks students find difficult or what common mistakes students make.

Based on these findings, the following three goals are set for the design of LabEX:

- **Provide lab staff with a summary of students' progress.** Lab staff always need to be aware of the overall progress of participating students; if there is an issue – such as a difficult task or unclear

description of problems – lab staff need to promptly respond within the limited lab session time.

- **Assist instructors spot students needing help.** Ideally, instructors must allocate their assistance to those students who are struggling more as opposed to those who ask many questions. Therefore, it is necessary to enable assistants to readily spot students who are struggling and need help.
- **Provide lab staff with a suitable way to perform code review.** The quality of code matters in programming and it would be ideal for lab staff to qualitatively review all students' codes. Since it is not feasible to read all code lines, LabEX provides an efficient tool to help assistants to review more code with less effort.

IV. Understanding Struggling Students

In programming lab sessions, it is important for TAs and instructor to identify struggling students and provide them with help so that they understand the material and complete their tasks. However, in large-scale lab sessions where participants are with a diverse level of programming skills and different cultural and personal threshold to ask for help, it is challenging for lab staff to identify truly struggling students.

Furthermore, it is not ideal to sightlessly help students immediately after they express any sign of struggle, because then they lose the chance to explore and learn by themselves. Hence instructors should give students sufficient time to solve problems independently; at the same time, they should spot struggling students and assist them before they lose interest and give up.

LabEX employs statistical methods to detect struggling students and measure their degree of struggle, so that instructors can efficiently help those struggling students.

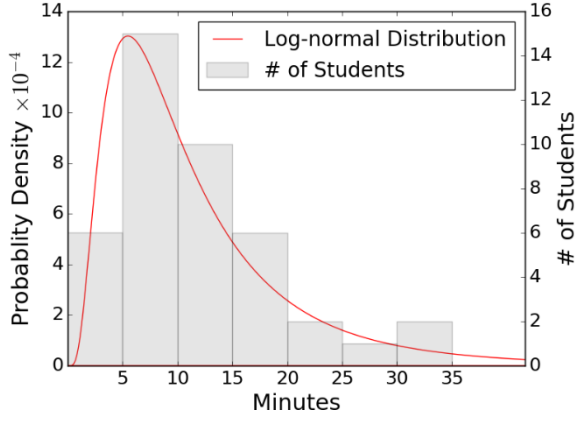
V. Estimating Distribution of Problem Solving Times

<i>Task</i>	<i>Avg. Time</i>	<i>SD</i>	<i>Min. Time</i>	<i>Max. Time</i>
1	6.3	4.5	1.0	20.0
2	9.2	8.0	2.0	34.0
3	4.8	6.2	2.0	25.0

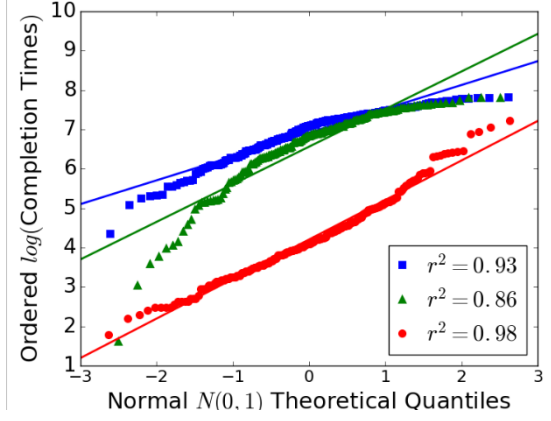
Table 1: Students' Task Completion Times in a Session (Min.)

One important feature that can express the level of student's struggle is the amount of time a student takes to complete a task. Therefore, LabEX utilizes task completion time as a major factor in finding struggling students. However, students spend an unequal amount of time to solve an individual problem in lab sessions. The time to complete a particular task depends on many factors such as a student's

understanding level, the required skills to solve the problem, and inherent difficulty of the problem. In our observation students spend as little as less than a minute, and as much as 34 minutes to solve a problem in the 1.5-hour lab sessions (See Table 1).



(a) A Sample Task



(b) Q-Q Plots

Figure 2: Assessment of Modeling Problem Solving Times with A Log-Normal Distribution; (a) shows the distribution of task completion times and the estimated log-normal distribution for a sample problem. (b) shows that the task completion times for sampled task follow log-normal distribution with the coefficient determination value (r^2 value) close to 1.

Although task completion times vary significantly with problems and students as shown in Table 1, we observed that for most problems, the completion times of all students closely follow log-normal distribution. Figure 2a shows the task completion times of students in a single lab session for one sampled problem. The vertical bars specify the number of students who solve the problem within a given time window; the red line represent the estimated log-normal distribution. We can see that the shape of the vertical bars and the lognormal distribution look very similar.

To verify that the task completion solving times follow log-normal distribution, we employ a quantile-quantile plot (Q-Q plot); we take the logarithm of the task completion times and plot the values in a Q-Q plot. Figure 2b shows the plots for three sampled problems; the plots for other problems are similar to one of the three plots. When we computed the coefficient of determination, or r^2 values, they are close to 1, indicating the data is linearly related to the model (log-normal distribution).

We observed some outliers; first and last few data points are plotted further away from the regression lines for some problems. Thus, when we estimate log-normal distribution from our observed data, we ignore first few data points if the estimated parameters using them are significantly different from the estimations using other data points. The outliers in the tail (last few data points) are of less significance

in our use case, because they do not need to be exactly modeled. The data points represent students taking exceedingly long time to complete a problem, and we aim to help those students before they spend too much time on a single problem.

Note that for identifying struggling students, a simple heuristic such as averaging the task completion times of first N students to use as a reference point cannot be used because it is unstable; the average value of ordered times varies significantly depending on the sample size N , which is hard to determine.

5.1 Estimating Parameters with Order Statistics

Ideally, we want to have the parameters of the distribution for the problem sets before lab sessions, so that we can identify difficult problems and struggling students in a timely manner.

While it is not possible to obtain the distribution parameters in advance, we can incrementally estimate the parameters as we get data points during a lab session by using the properties of order statistics [2]. That is, we examine the completion times of first k students for each problem to estimate the distribution parameters of the problem. It should be noted that naively estimating the parameters with k samples based on maximum likelihood estimation is incorrect because the k samples are biased. The k data points are not uniformly sampled, but they are smallest k samples, hence, correlated with each other.

To take the sampling bias into account we use order statistics to estimate the distribution parameters. Given a probability distribution X and n random samples from X , the k th order statistics of n samples is the distribution of the k^{th} minimum value in the n samples. For our estimation we model k^{th} student's completion time as a sample drawn from k^{th} order statistics of n samples (where n is total number of students in a lab), rather than a sample drawn from distribution X itself.

In this formulation where k^{th} student's time x_k follows distribution $X_{k,n}$ (the k^{th} order statistics of n samples from distribution X), the maximum likelihood estimation of parameter μ and σ of X can be obtained as following: $\argmax_{\mu, \sigma} P(X_{1,n} = x_1, X_{2,n} = x_2, \dots, X_{k,n} = x_k; \mu, \sigma)$. To accurately estimate the parameters to maximize the likelihood, it requires running computationally expensive algorithms such as EM [13]; this is ineffective since we need the estimations in real-time.

Instead of maximizing the above likelihood to get an accurate estimation, we compute MLE for individual data points to obtain multiple candidate estimations; then we select the one that best fits for

all k data points.¹ This heuristic is efficient when applied in real-time and gives reasonably good estimation for our purpose. In order to obtain candidate estimation values, we compute MLE for the order statistics for standard log-normal distribution with Monte-Carlo simulation beforehand. Let the precomputed MLE of i^{th} order statistics of n samples be o_i ; hence we have o_1, o_2, \dots, o_k . Then we compute μ and σ with the completion time x_i and o_i using the equation $\ln(o_i) = (\ln(x_i) - \mu)/\sigma$. Since there are two unknowns, two data points are required; we randomly select two data points ($\{x_i, o_i\}, \{x_j, o_j\}$) to compute the values. Because there may exist some outliers in first few data points, we exclude them in the estimation if the parameters computed with those data points are significantly different from other estimated values.

After collecting sufficient number of candidate estimations of μ and σ , we verify how well they describe the k data points. The candidate parameters are applied to the likelihood function [14]: $L(\mu, \sigma) = \prod_{i=1}^k f_x(x_i; \mu, \sigma) \times (1 - F_x(x_k; \mu, \sigma))^{n-k}$ where f_x and F_x denote pdf and cdf of log-normal distribution. Then the parameter that gives the highest likelihood is selected to determine the final distribution.

The distributions of problem completion time that we estimate can be used for multiple purposes. In LabEX we mainly use them in two ways; to infer the difficulty of problems and identify struggling students.

5.2 Modeling Students Struggle

To model struggling students and their degree of struggle, we further monitor the interactions of students while they work on lab session tasks. Among many interactions that can be monitored we focus on the following two: the frequency of running the solutions for each task and the amount of code changes (measured in key strokes) between the code executions. Although we also monitor other values such as the number of total runs for a task, we discovered in lab session trials that the two values – frequency of runs and code changes between runs – are most effective in representing struggling students. This is on par with our findings in lab observations; we noticed that students, when struggling, often repeat running their code, try (many) minor and trivial code changes, or just idle away.

While task completion times follow a log-normal distribution, the above two measures are well described by a normal distribution. With the three measures we identify struggling students for a task

¹This heuristic is similar to the technique presented in [20]

in the following way. After we estimate the distribution of the task completion time, we select a threshold time that pinpoints bottom X% (30% by default) students in the distribution; we only consider the bottom 30% as potentially struggling students. For those, we define the degree of struggle using the three measures – the time spent on the task, frequency of runs, and amount of code change between runs. The values of the three measures are first normalized to standard normal/log-normal distributions; then we compute the distance of how far away the three values are from the average of the distributions. The degree of struggle is a weighted sum of those distances. We compute the degrees for all the tasks for struggling students and sort them by their degrees to help lab staff to identify those who need most help.

VI. Qualitative Code Review

6.1 Clustering Students' Solution

Merely completing a task (by passing unit tests) does not guarantee that a student has learned the concepts well, especially in programming exercises where students can come up with a variety of solutions. Plenty of undesirable solutions are usually found, in our experience, among all answers to just a single problem. Oftentimes these undesirable solutions are left unnoticed during lab sessions, and students leave the sessions not having a chance to receive feedback and improve their code.

Identifying undesirable solutions during lab sessions is challenging because an instructor has to review all the students' codes which is time consuming. In one 100-minute lab session in our institute, students wrote 8,253 lines of code in total in a single session excluding intermediate solutions. To review all the solutions the instructor must check more than one line of code every second. This is clearly impossible.

In the following, we demonstrate an example of an undesirable solution, which was written by a student in a lab session.

```
1:  def isPrime(number):
2:      if number == 2: return True
3:      for i in range(2,number):
4:          if (number/i - number//i) == 0:
5:              break
6:          elif (number/i - number//i) !=0:
7:              if i == number - 1:
8:                  return True
9:              continue
10:     return False
```

Although the code returns correct answers for all the positive integer inputs, the code is generally considered poor because of its style. His solution reveals that 1) he is not aware of the modulo operator in Python, 2) he does not fully understand how if-statement works (unnecessary check in line 6), 3) he is confused about loop controls (unnecessary continue statement in line 9), and 4) he does not know making special cases is not a good style (line 2).

To enable effective and timely feedback, LabEX makes reviewing process easy with a code clustering technique. There have been many works in detecting/clustering similar code in the context of software maintenance and plagiarism detection; in LabEX we apply the technique in the context of real-time code review. Thus, our clustering algorithm is less accurate but much more lightweight and suitable for real-time usage.

Algorithm 1 compute_cluster(codes, threshold)

```

1: for  $c_i$  in codes do
2:    $cluster_i = \{i\}$ 
3:    $sig_i = \text{make\_signature}(\text{get\_ast}(c_i))$ 
4:  $Q = \emptyset$ 
5: for all  $sig_i, sig_j$  do
6:    $dist_{i,j} = \text{compute\_dist}(sig_i, sig_j)$ 
7:    $Q \leftarrow \{dist_{i,j}, i, j\}$ 
8: while  $Q \neq \emptyset$  do
9:    $d, i, j \leftarrow Q.min()$ 
10:  if  $d > \text{threshold}$  then
11:    break
12:   $\text{merge}(cluster_i, cluster_j)$ 

```

Algorithm 1: LabEX's Code Clustering Algorithm

Algorithm 1 describes the clustering algorithm in LabEX. Our algorithm is along the same lines of AST (Abstract Syntax Tree) fingerprinting algorithms [14] [15], but much simpler and lightweight. Our algorithm is based on hierarchical clustering [16], thus it can simply control the granularity of the clustering. For each solution code, the algorithm creates a signature of the code with its AST. Each AST node is encoded into one-byte character; the specific names of variables and the values of constants are ignored. Then the distances – edit distance [17] by default, but tree edit distance [18] can be used as well – among all the signatures are computed. The computed distances are sorted and retrieved to merge the two clusters containing the corresponding code signatures. This is repeated while the retrieved distance is smaller than a given threshold, or all the distances are retrieved.

Although the algorithm is not as accurate or robust as other AST fingerprinting algorithms, it is lightweight and effective in our context. The first two steps of the algorithm – creating signatures and computing distances – can be run incrementally as a new solution is submitted. Hence the clustering

effectively runs fast by only computing the last step of the algorithm. Having prompt response time is important in our use case because a lab instructor should be able to quickly adjust the clustering parameters as he reviews solution code.

6.2 Comparing Code Clustering Algorithms

To test if our clustering algorithm serves well our purpose, we have evaluated three code clustering algorithms – our algorithm in LabEX, OverCode [9], and CloneDigger [19]. For the evaluation, we have collected the solutions of students for three problems in a lab session; we hired seven experienced programmers to score the pair-wise similarity scores of the solutions in 5-point scale. Then we measure the quality of the created clusters of the three algorithms with the following metric that computes pairwise similarity score sum (PSS).

$$PSS(C) = \frac{1}{k} \sum_{k=1}^K \frac{2}{|C_k|(|C_k| - 1)} \sum_{x, y \in C_k} score(x, y) - \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{k'=k+1}^K \sum_{x \in C_k} \sum_{y \in C_{k'}} score(x, y)$$

The function averages the similarity scores of codes in same clusters and subtracts the average similarity scores of codes in different clusters. Table 2 compares the average PSS values over the three problems for the clustering algorithms. The total number of student solutions is 182. Clearly the clustering algorithm of LabEX achieves the highest score in PSS, which implies that the clusters of our algorithm is most similar to how an experienced programmer would cluster the solutions among the compared algorithms.

Algorithm	LabEX	OverCode	CloneDigger
Avg. PSS	0.29	-2.50	-1.70
Avg. Exec. Time	0.017	6.612	6.720

Table 2: Evaluation of Clustering Algorithms.

For the execution times, our clustering algorithm is fastest among the algorithms. LabEX needs to repeatedly run clustering algorithm in real-time, as an instructor adjusts the clustering granularity to re-cluster solutions for code review; hence the execution time is important. In our experience, lab staff were reluctant to use the code review interface if it takes more than a couple of seconds for the clustering. While the execution times of our algorithm are less than 0.1 second, OverCode takes 8.3 seconds and CloneDigger takes 13.8 seconds to run in the worst case. We believe that the execution time of 8.3 seconds is too slow to be used in our context.

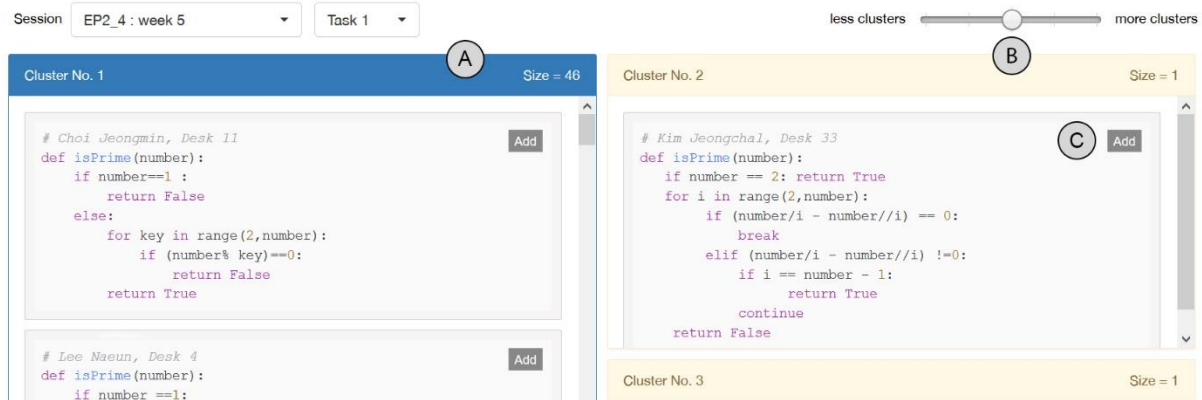


Figure 3: Snapshot of Code Reviewing Interface (Student names are anonymized.): (A) displays a cluster of solutions, clusters of small size have yellow headings. (B) shows a slider bar, by which the granularity of the clustering can be adjusted. (C) presents the button that once clicked, adds the code to the official solution.

6.3 Reviewing with Clusters

In code review interface, instructors simply select the lab task to review the students' code, which are displayed after being clustered by their similarities. Figure 3 shows the code review interface that displays three code clusters in the text areas. The clusters are sorted by their sizes; if its size is below a threshold (five by default) the cluster is considered as an outlier and its header is colored in yellow. Solution code within each cluster can be scrolled and reviewed.

As the instructor reads solution code, he may feel that the clustering is too fine-grained or too coarse-grained. In such cases he can slide the knob on the top-right to review code in more or less number of clusters. In our experience, the number of clusters for a task is as little as just one or two, and as many as one-third of class size. We observed that having too many clusters often indicate that the students are exploring in too many directions and need some guideline.

The solution code is tagged with student's name in order for the lab staff to easily locate the student and give feedback on his code. Each solution code also has a button which, when clicked on, adds the code to the verified solutions list that will be given to the students after the lab session.

VII. LabEX User Interface

Having presented the major components of LabEX, now we describe how all the functionalities are combined to form an intuitive user interface.

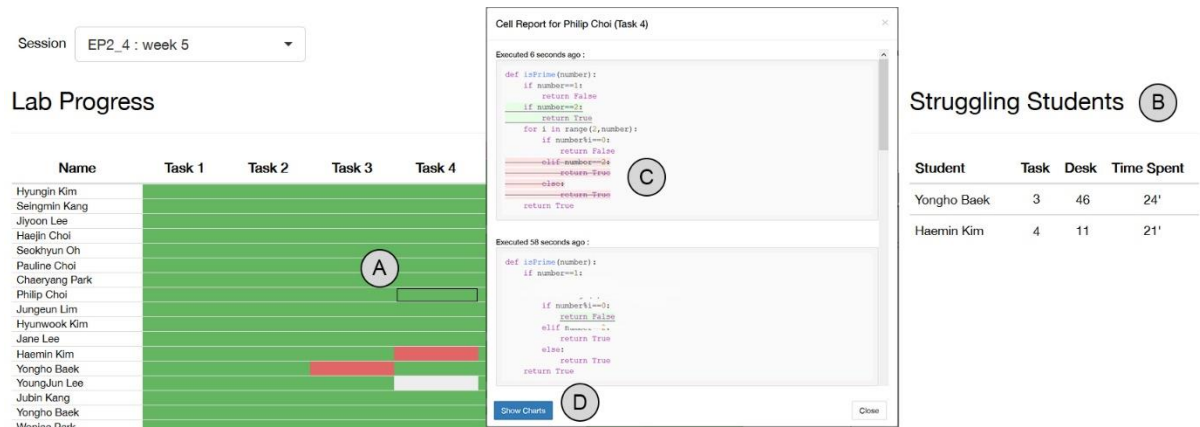


Figure 4: Snapshot of Progress Overview (Student names are anonymized.): (A) displays a cell representing a students’ status on a task. (B) displays a list of struggling students and the corresponding tasks. (C) shows a student’s solution code; it also shows previous versions with highlighted diffs. (D), once clicked, shows additional statistics such as time spent on the task.

7.1 Students Interface

As an extension of Jupyter Notebook, LabEX provides a notebook-like interface to students. Using a web browser, students interactively edit and run their code embedded in a document, which provides task descriptions, videos, and links to background information. In LabEX, students can easily retrieve their previous trials, which is useful to students as well as lab staff when they help them. Interactions of students in web browsers are recorded in the server for real-time analysis.

7.2 Lab Staff Interface

students’ progress in real-time. LabEX has two main interfaces – one to monitor overall students’ progress and another to review student solution code. Figure 4 shows the interface to monitor the overall progress. The progress is shown for each student and each task; a small box representing a task is marked as green if the student completes the task, gray if he is working on it, white if he has not started on the task, and red if he is having a difficulty (according to our model described in Section 4). The progress is sorted so that the student who completed most number of tasks is placed at the top. This helps lab staff to quickly figure out the overall progress in a single look; it also makes it easy to find the tasks students are stuck in or understand the performance skew among students.

Lab staff may click on an individual box representing a particular student and task to see the solution code of the student for the task. It retrieves his most recent solution as well as his previous trials since they are often useful in understanding the misconceptions students might have.

On the right side we list struggling students that LabEX identifies based on the statistical model. The list is ranked by the degree of struggle, and it contains the name of the student, his desk number, the task id, and the time spent on the task. The list entry can be clicked on to review the student's current and previous solution codes.

Another main interface in LabEX is the code review page. The page, as shown in Figure 3 displays the solutions of students for a selected task. As described previously, similar solutions are clustered so that an instructor can effectively review the solution code. The page provides a sliding bar for adjusting the granularity of the clusters, so an instructor can group solutions more (or less) aggressively as necessary.

The code review interface can be used during lab sessions to identify poor solutions and give real-time feedback to the students. Students appreciated such real-time feedback and liked the opportunity to improve their solution. Moreover, the interface helps find creative/interesting solutions so that they can be shared with the rest of the students during or after lab sessions. If an instructor decides to include student's code in the verified solutions list, he simply clicks on a button next to the student's code.

7.3 Mobile View

In offline lab sessions, lab staff rarely sit in front of computers; they usually walk around to check with students and help them. If they have to visit on a desktop computer for the lab status, they are likely to miss important issues. Thus, it is essential for lab staff to be able to view important summary of lab status on their mobile devices.

LabEX provides mobile-friendly interfaces for small mobile screens. Especially, LabEX compiles the list of struggling students and make it available to view on mobile devices and check their status. With this view, lab staff can quickly check on struggling students to provide them with assistance.

VIII. Evaluation

We evaluated LabEX with a case study and real-world experiment in a large-scale programming course in our institute.

8.1 Case Study

To quantitatively evaluate LabEX, we performed a small case study; we gathered 55 freshmen attending the course, and randomly divided them into two groups to run two separate lab sessions with and without LabEX.

The two groups (Group A and Group B) have 28 and 27 students each. Our experimental lab sessions were 50 minutes long, and students were given the same problem set consisting of four problems. Three (same) TAs helped the two groups of students in the two sessions. The participating students have three months to four years of programming experience, except for a student with ten years of experience; half of the students have six months of programming experience.

The session for Group A was run without LabEX; in Group B's session the TAs utilized LabEX to identify struggling students and help them. Before the session, we told the students in Group B that we would be monitoring their progress in LabEX and help them as necessary; we also suggested the students to try to limit their questions below four if possible. We did not have such request for Group A because in our previous experience, such a policy (without LabEX) made TAs not be able to properly help students, hence made many students underachieve in lab sessions.

Figure 1 shows the amount of time the TAs helped each student; Figure 1a represents Group A and Figure 1b represents Group B. Horizontal axis is for elapsed time in minutes, and vertical axis shows the amount of time each student is helped by a TA in a 10-minute window centered at the current elapsed time. A color in the chart represent a student in the lab session. The stacked bar charts at the top, show the total time TAs spent assisting each student.

We can immediately see that in Group A, the TAs spent significantly more amount of time to help students compared to Group B; for Group A, TAs spent 81 minutes in total in helping the students and for Group B, they spent 48 minutes. Also, in Group A, much of TAs' time is monopolized by a single student, who kept asking many simple questions that he could have figured out by himself.

However, the students in Group B performed as well as, if not better, than the students in Group A. The students in Group A successfully completed 1.18 problems on average, and the students in Group B completed 1.26 problems. If we exclude the exceptional student in Group A, who had ten years of programming experience and previously won two medals in local programming competitions, the students in Group A completed 1.07 problems on average.

To test if the improvements are statistically significant, we perform a t-test. In Group A and Group B, 16 and 20 students are assisted by the TAs. The mean time spent for each student in Group B is 2.4 (min.) and the standard deviation (SD) is 1.39. In Group A the mean and SD are 5.06 and 5.26 respectively, and the two-tailed p-value equals 0.0365. By conventional criteria, these numbers demonstrate that there is a significant difference between the two groups. That is, TAs spent less time to help students in Group B with LabEX compared to Group A, even though students in Group B achieved as well as students in Group A. Furthermore, we perform f-test and obtained F-statistic value of 14.28. This leads us to conclude that the variances in assisted time for each student in Groups A and B are significantly different with p-value less than 0.0001; i.e., students in Group A are unevenly helped by the TAs, while students in Group B are more evenly helped by the TAs.

From this analysis, we concluded that TAs spend their time more efficiently, if they are assisted with LabEX; we believe that TAs can use the saved time in improving the quality of the lab session by, for example, creating hints for some problems or making more difficult problems for advanced students.

8.2 Real-World Experiment

We applied LabEX in a large entry-level programming class in our institute. The class teaches basic concepts in programming using Python language and it is taken by 272 students, most of who are freshmen. The students in the class are divided into six groups and assigned to one of six 100-minute lab sessions given every week; thus, there are roughly 45 students in each lab session. There were two to three TAs in each lab session, half of whom were undergraduate students and the rest were graduate students.

8.3 Methodology

In collaboration with the lab instructor and TAs, we adopted LabEX in the lab sessions for six weeks in total. In the first two weeks we disabled the monitoring interface for comparison and only traced students' interactions. For the remainder four weeks we enabled the interface and encouraged the lab staff to use the interface. We watched all the lab sessions that LabEX was used in; we attended 36 sessions in total and stayed for the entire sessions in most cases.

The lab sessions are run in a naturalistic way; there was no particular tasks for lab staff or students; and we mostly observed the lab staff, the students, and their interactions. A couple of times, we gave suggestions to lab staff based on our observations after lab sessions.

At the end of our study, we conducted 15-minute informal interviews with the lab staff and a few selected students. In the interviews we asked them to discuss some of their interactions we observed during the lab sessions. They also stated their opinion on their experiences using LabEX and suggested possible improvements.

8.4 Qualitative Observations

As in our case study, we noticed that a number of students take up TAs time, asking many trivial questions or try to get as many hints as possible; we realized that not only do those students make little effort to complete the tasks, but they also monopolize the instructors' time. After having noticed it in the first couple of sessions, we suggested to the lab instructor to limit the number of questions a student may ask during a lab session. This scheme, however, did not work well in the first two weeks because (without LabEX) the lab staff were not aware of how students are doing and limiting the number of questions made it harder for them to grasp what is going on in the lab. Hence after a few sessions, they let the students to freely ask questions again.

After we enabled LabEX monitoring interface in the third week, we suggested to try the scheme once again and limit the number of questions in the lab sessions. The lab instructor was hesitant in the beginning, but when the scheme was applied, it turned out to be very successful. With the progress summary in LabEX the lab staff were fully aware of the overall progress and could identify struggling students to help them; the instructor and TAs seemed comfortable running the lab sessions. The instructor and TAs saved much time therefore found the chance to review student solutions, discuss their findings in the lab, improve lab tasks, and think of more challenging tasks.

We also observed that the instructor quickly adapted to employing LabEX in various lab situations. When LabEX informed that a particular task seemed too difficult based on the estimated distribution of the task completion time, he gave a short lecture (or hint) for the task after reviewing a few corresponding completed solutions using the code reviewing interface. After a few similar occurrences, the instructor began to take advantage of the progress summary to decide when to give hints for a task.

The code review interface was appreciated and frequently used in the lab sessions. Mostly, the interface helped discover poor solutions, but it also assisted to find intriguing solutions. In one occasion a student utilized Python's list comprehensions in his answer, a rather advanced syntax for entry-level learners. Having acknowledged that he knew more advanced topics, the lab staff were able to assign the student more challenging tasks, giving him the chance to further improve his skills. In another occasion, a

student submitted a wrong solution for a task to implement *find_min()* function in a binary search tree. Although his solution did not properly handle a recursive case, it passed all the given tests because the test codes did not fully cover corner cases. After learning this, the instructor updated the test codes and pointed out the issue to the student. The student appreciated the prompt feedback and fixed his code. One TA, who taught in the lab class for two semesters in a row, was particularly interested in LabEX system and suggested several improvements. For example, he asked to display students' names when hovering over a certain part of the progress screen or asked to use a particular syntax highlight style in the code review interface.

8.5 Post-Study Interview

After the end of all the lab sessions, the lab instructor and TAs participated in an informal 15-minute post-interview. The questions were designed to evaluate LabEX's impact on elevating lab sessions' efficiency to help students with conceptual learning and skill acquisition. They all recognized the distinct improvement of lab sessions brought by LabEX since it saved them time and helped them to reach out to students in real need for help. TA1 commented, "I could spot students who had a hard time with a task and were reluctant to ask a question because they either were shy or had given up.". TA2 said, "Approaching struggling learners proactively to help them cheered them up and increased their determination to solve the problems".

The lab staff commonly stated that they appreciate the code clustering interface. They stressed that the tool enabled them to evaluate students code qualitatively, which they could not do previously for its timely process. Some of them found it to be fun to go over "outliers" to find poor-quality solutions and discuss it with the students who wrote them.

TA3 pointed out that it was helpful to be able to go over previously tried solutions of a student. When asked why he reviewed a student's solution code (current and previous tries) before helping him, he replied, "I wanted to understand his line of thoughts and check if he knows the concepts."

We observed that undergraduate TAs, compared to graduate TAs, are generally more passionate about discussing their experience of using LabEX and helping students. Possibly it is because they participated in the same lab class just a few semesters ago; they might have felt frustration and embarrassment previously in the lab sessions and may want to help students in similar situations.

8.6 Discussions and Limitations

From our real-world evaluation, we showed that LabEX is useful and improves the quality of programming lab sessions. We observed noticeable difference in the way lab staff interact with students after applying LabEX. All the lab staff enjoyed using the system and helping students that need more help.

A few TAs expressed their concern regarding the scalability of LabEX in terms of its visual interface. They mentioned that the progress summary interface may not be able to effectively display the progress of very large number (like 1,000 or more) of students. While we believe that such an extremely large lab class is not ideal, we could virtually split the lab and assign them to multiple LabEX sessions.

Another concern is that LabEX would not be as effective in more advanced-level programming courses. We do not have experience of applying LabEX for advanced level courses, and its efficacy may be different for those courses.

IX. Conclusion

Programming lab sessions give students hands-on experience and enable them to improve their skills through guidance and feedback from instructors. In large-scale lab sessions, however, it is challenging for lab instructors to be fully aware of how individual student is learning, hence the instructors may not be able to give necessary guidance and feedback to all students.

This dissertation presents LabEX, a system for real-time monitoring of how students learn in programming lab sessions. LabEX helps instructors to comprehend the overall status of all students as well as the pace of individual students. Using statistical models, LabEX identifies struggling students and draws instructors' attention to them. LabEX applies a code clustering algorithm for real-time code review, which enables lab instructors to detect undesirable solution codes and provide necessary feedback and programming advice.

We evaluated LabEX in a large programming course attended by 272 students. LabEX was used by 17 lab staff for six weeks and 36 sessions. All the lab staff found LabEX to be helpful and used it throughout the lab sessions. Our study confirmed that LabEX noticeably improved the quality of lab sessions for lab staff as well as the students attending the sessions.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my ex-advisor Prof. Jiwon Seo for the continuous support of my studies and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I would like to thank the rest of my thesis committee: Prof. Sam H. Noh and Prof. Sungahn Ko, for their encouragement, insightful comments, and questions.

I thank my fellow lab mates and friends: Vicente Bolea, Ali Tousi, Junghyun Kim, Jinwon Lee, Bongki Cho, and Zumrat, for their assistance throughout the evaluation of LabEX and for all the fun we have had in the last two years. Also, I thank Mr. ChangJoo Oh for his help throughout the live-testing of LabEX in programming lab sessions.

Last but not the least, I would like to thank my family: my mother Safieh Aghamirkarimi and my wife Marcela, for their enormous support throughout my life.

REFERENCES

- [1] N. . Titterton, C. M. Lewis and M. J. Clancy, "Experiences with Lab-Centric Instruction.," *Computer Science Education*, vol. 20, no. 2, pp. 79-102, 2010.
- [2] H. A. David, *Order Statistics*, Wiley, 2003.
- [3] N. B. Sarter and D. D. Woods, "SITUATION AWARENESS: A CRITICAL BUT ILL-DEFINED PHENOMENON.," *The International Journal of Aviation Psychology*, vol. 1, no. 1, pp. 45-57, 1991.
- [4] P. J. Guo, "Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 2015.
- [5] C. . Coffrin, L. . Corrin, P. . de Barba and G. . Kennedy, "Visualizing patterns of student engagement and performance in MOOCs," *Proceedings of the Fourth International Conference on Learning Analytics And Knowledge*, vol. , no. , p. , 2014.
- [6] P. J. Guo and K. Reinecke, "Demographic differences in how students navigate through MOOCs," in *Proceedings of the first ACM conference on Learning @ scale conference*, 2014.
- [7] R. F. Kizilcec, C. Piech and E. Schneider, "Deconstructing disengagement: analyzing learner subpopulations in massive open online courses," in *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, 2013.
- [8] Y. Chen, Q. Chen and M. Zhao, "DropoutSeer: Visualizing learning patterns in Massive Open Online Courses for dropout reasoning and prediction," in *Visual Analytics Science and Technology (VAST)*, 2016.
- [9] E. L. Glassman, J. Scott, R. Singh, P. J. Guo and R. C. Miller, "OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale," *ACM Transactions on Computer-Human Interaction (TOCHI) - Special Issue on Online Learning at Scale*, vol. 22, no. 2, 2015.
- [10] J. Huang, C. Piech, A. Nguyen and L. J. Guibas, "Syntactic and functional variability of a million code submissions in a machine learning mooc," in *AIED Workshops Proceedings Volume*, 2013.
- [11] "Project Jupyter," [Online]. Available: <http://jupyter.org/>. [Accessed September 2016].
- [12] "Jupyterhub," 2016. [Online]. Available: <http://github.com/jupyterhub>. [Accessed September 2016].
- [13] H. Ng, P. Chan and N. Balakrishnan, "Estimation of parameters from progressively censored data using EM algorithm," *Computational Statistics & Data Analysis*, vol. 39, no. 4, pp. 371-386, 2002.
- [14] M. Chilowicz, E. Duris and G. Roussel, "Syntax tree fingerprinting for source code similarity detection," in *IEEE 17th International Conference on Program Comprehension*, 2009.
- [15] B. Hummel, E. Juergens, L. Heinemann and M. Conradt, "Index-based code clone detection: incremental, distributed, scalable," in *IEEE International Conference on Software Maintenance*,

2010.

- [16] P. J. Rousseeuw, Finding groups in data: an introduction to cluster analysis, John Wiley & Sons, 2009.
- [17] K. Kukich, "Techniques for automatically correcting words in text," *ACM Computing Surveys (CSUR)*, vol. 24, no. 4, pp. 377-439, 1992.
- [18] P. . Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol. 337, no. , pp. 217-239, 2005.
- [19] P. Bulychiev and M. Minea, *Duplicate code detection using anti-unification.*, Spring Young Researchers Colloquium on Software Engineering, 2008.
- [20] G. Kumar, G. Ananthanarayanan, R. Sylvia and I. Stoica, ""Hold'em or fold'em?: aggregation queries under performance variations," in *Proceedings of the Eleventh European Conference on Computer Systems. ACM*, 2016.