Badkan

Dvir Reut [✡] Bismuth Samuel [❄]

Advisor: Dr Azaria Amos

July 2, 2019

Abstract

Badkan is a server for automatic checking and grading of programming assignments. There exist several ways to grade an assignment, on this paper, we will discuss the technologies used by the Badkan, and how it grades assignment.
There are two type user on the Badkan: the students and the instructors. Given an exercise from any instructor, students must be able to submit an answer in the platform and get grade automatically. The simplest way to give a grade to any student is to run the code provided by the student with a specific input and compare the output of the student with the expectation the instructor had for the output. If the expectation correspond to the output of the student's code, then some point will be give to the student.
Another interesting way to grade student is by using "peer to peer grading": on this process, students grade them-self. Indeed, the first part of the process is to let student implement test-cases for the given exercise. Then, the second part is to let them submit their answer for the exercise. At this point, the platform is able to grade the test-cases submission and also the solution of the exercise. On this paper we will focus on how the process goes, and how the final grades are attributed.

[✡] Student of Computer Science (third year), Ariel University, Ariel 40700, Israel. Id : 204121446. Email: reutdvir3@gmail.com

[❄]Student of Computer Science, Ariel University (third year), Ariel 40700, Israel. Id : 342533064. Email: samuelbismuth101@gmail.com

1 Introduction

Peer to peer grading (P2PG) is the most relevant term to speak about the process presented in this paper. In a peer to peer network, the "peers" are computer systems which are connected to each other via the Internet. Files can be shared directly between systems on the network **without the need of a central server**. Regarding the peer to peer grading, the central server can refer to the instructor. That is, when speaking about the P2PG, the main challenge of the process is to grade students without help of any instructor.

How can the students grade them-self such that the final grading remain fair and truthful?. From the course Algorithm 2 taught by Dr Erel Segal-Halevi: A mechanism is said to be truthful if a agent always maximizes his utility by declaring his true input, regardless of what the other agents declare.

This paper approach use Unit Testing (UT). UT is a level of software testing where individual units / components of a software are tested. The purpose is to validate that each unit of the software performs as designed. That is, the process includes for main phases. The UT phase, the code phase, the conflict phase, and the grading phase.

2 Related work

An interesting paper [1] introduce the Crowdsourcing method. This method is not only use for the P2PG purpose, but also to classifying website content, or gathering data about the real world. In all the cases the goal remains the same: selecting workers with the best abilities for the task, and getting them to invest their best effort to obtain the most accurate answers. To overcome this problem, only worker that provide work get payment. And the payment is proportional to the provided work.

When using the Crowdsourcing method for the peer grading purpose, workers become students that need to grade their own assignments. Of course, the method have to care about the truthfulness of each students. Since all the students are submitting the same assignment the method want to reward student with they work agree. That is, a student that want to make a false declaration is susceptible to not being truthful and then, to manipulate his answer to reward more. Thus, the use of the Peer Truth Serum for Crowdsourcing (PTSC) is proposed. The PTSC makes the utility of each student depends on the effort invested in obtaining the answer do a given task. And contrary to other mechanism, there is no dependencies between two direct students. Therefore, the answer of the student A will not imply on the grade of the student B, but only the set of the answers (except the answer of student B) will imply on the grade of the student B. The algorithm 1 compute the PTSC score 1. Once the score computed, a normalization is made and the assignment are graded.

```
for student w that solves task $t_w$ do
    (1) Let $x_w$ = student's w answer.
    (2) Let $R_w$ being the ratio between the amount of $num(x_w)$ with the
      total number of answers.
    (3) Select a student d that solves $t_w$.
    (4) w is rewarded according the the next formula:
    $t(x_w, x_d) = \alpha \cdot (t_0(x_w, x_d) - 1)$ where $x_d$ is student's d answer, $\alpha$ is
      greater that 0 and $t_0$ is defined as
        $t_0(x_w, x_d) = \left\{ \begin{array}{ll} \frac{1}{R_w}, & \text{if } x_w = x_d \\ 0, & \text{if } x_w \neq x_d \end{array} \right\}$
end
```

Algorithm 1: Computing the PTSC score

Another paper [2] takes a probabilistic approach to compute the grades. Over 63, 000 peer grades are analyzed by three probabilistic models presented on the article. Prior to speak about the probabilistic models, let define what are the latent variables.

- True scores: is the the score considered as true for a given assignment.
- Grader biases: Every grader is graded himself by the grader biases. This index indicate either the grader has a big weight or not.
- Grader reliabilities: This number reflect how close the grade for the assignments are close to the true score.
- Observed grades: This is the grade attributed by the model.

The first model so-called "Grader bias and reliability" puts prior distributions over the latent variables and assumes for example that while an individual grader's bias may be nonzero, the average bias of many graders is zero. Thus, a distribution is given for each latent variable. The Gamma distribution for the True scores and some Normal distribution for the other variables.

"Temporal coherence" is the second model. This model is similar to the first model, except that it requires that we normalize grades across different homework assignments to a consistent scale.

Finally, the third and last model is called "Coupled grader score and reliability". This model accentuate the fact that if a grader is graduate with a high grade, his weight as a grader is increased. That is, some new variables are added to compute the weight of every grader according to his grade.

3 The P2PG system

3.1 Background

The system includes two types of users: the students and the instructors. Each student will enroll a tester behavior (writing and submitting test cases) and a coder behavior (writing and submitting code project).

There are two main independent problems:

- Recognize when a test is a *wrong test.*
- Attribute grade to the students.

Definition 1 (Wrong test) *Test with a non-sense: like expectTrue(1+1==3).*

Prior to present the whole system, let define two critical axioms.

Axiom 1 *The system want to avoid a tester that writes a lot of test cases but not necessarily relevant test cases.*

Axiom 2 *The system want to avoid a coder that write a solution that failed on his test cases.*

The P2PG system is divide into four main phases. Each phase has it own deadline, and each deadline of a phase is the beginning of the next phase.

3.2 The UT phase

During this phase, each students take the role of tester. That is, given the assignment, the student submits unit tests. The student is able to submit his tests numerous time, such that only the last submission is keep. Every time that the student will submit, a feedback is displayed. The feedback includes only compilation error. If the tests compile right, the system also notify the student that everything works fine.

3.3 The code phase

The code phase is the time for the students to submit their solutions for the assignment. Recall that the system stored several unit tests. That is, every time a student submits a solution, the system run every unit tests for the given solution. The student can similarly to the UT phase, submit several time. As feedback, the student see the ratio *number of test passed / total number of test* displayed on his screen. And for each test failed, the code of the test is displayed with a check box "wrong test". By checking the button "wrong test", the student claim that the failed test is a *wrong test.* That why the system needs a conflict phase to handle all the wrong tests.

3.4 The conflict phase

Recall that some test are reporting being wrong. During this phase, the system is going to decide either the test is actually a wrong one, or not. To do this, the system first send to each tester all reporting test he wrote. That is, if the tester agreed with the report, the test if officially declare wrong and the test is deleted from the UT. If the tester disagree with the charge, the system ask the instructor to decide if the test is wrong or not. Given x charging UT, the instructor can decide to automatically declare all the test as wrong, or as good. He can also look every test and manually decide if the test is wrong.

3.5 The grading phase

3.5.1 Data

At this stage, the system recorded some data that is used to compute the final grades.

Define $S = \{s_1, s_2, \cdot\cdot\cdot, s_n\}$ the set of students. For each student define $TS_i = \{ts_{i1}, ts_{i2}, \cdot\cdot\cdot, ts_{im}\}$ the set of test implemented by the student i. So there exist $n - 1$ TS arrays. Notice that each test is different (there is no possibility for two students to implement the same test case). The current data is composed of n the solution table.This table is shared by all the students and is universal.

	ts_{11}	ts_{12}	...	ts_{21}	ts_{22}	...	ts_{nm}
s_1	PASSED	FAILED	...	PASSED	PASSED	...	FAILED
s_2	FAILED	PASSED	...	FAILED	FAILED	...	FAILED
...	...	...	...	...	...	...	...
s_n	FAILED	FAILED	...	FAILED	PASSED	...	PASSED

Table 1: Solutions table

3.5.2 Requirements

The grading phase is the most tricky phase. Let first understand the main problems and the solutions.

Theorem 1 (Good test theorem) *On this phase, it is known that every single test is not a wrong test.*

Proof 1 *Automatically derives from the conflict phase.*

Derivative problems: (by *priority*)

1. Force the student to write “good” test cases.

2. Force the student to write “good” solutions.

3. Grade the students for their solutions (of course the grade has to be logic).
4. Grade the students for their tests (of course the grade has to be logic).
5. Bother student at least as possible about something else that "coding".
6. Bother instructor at least as possible.

Remark 1 (priority) *The priority can be customize by the instructor only. To customize priority, the instructor will have to choose between some options.*

3.5.3 Solutions

1 Grade the test submission: Assuming that the grade is actually higher for a good test writer, the student will be forced to write "good" test cases.

2 Grade the solutions submission. Similar to 1.

Grading the solutions - 3 The system can proceed by two ways (the instructor has to choose one): If there are t tests do: weight=$\frac{100}{t}$. For each passed test, give to the solution submitter weight points. For each failed test, give to the solution 0 point. In this case, the instructor will have to input a parameter lambda between 0 and 1, where 0 gives more points when passing an easy test and 1 gives more point to the student that pass a hard test. Assuming there are s students, such that student s_i is a *better tester* than student $s_i + 1, \forall i$. Then, compute the partition algorithm, such that the algorithm return the array "partition array". Thus, given a student solution such that the var grade is initialized at 0, do:

```
if student pass $test_i$ designed by student $s_j$ then
    grade += partition array[j] $\cdot \frac{1}{numberoftestdesignedbyj} \cdot 100$
else
    pass
end
```

Algorithm 2: Compute the Solution grades

Definition 2 (better tester) *The student that implements tests such that the sum of the students that failed at his tests is the highest.*

Assuming that for each failed test, the function f is defined as:

$$f(ts_{11}) = \left\{ \begin{array}{ll} 1, & \text{if } ts_{11}, ispassed \\ 0, & \text{if } ts_{11}, isfailed \end{array} \right\}$$

such that ts_{11} is a arbitrary test.

Given two student i and j s.t i $\neq$ j, we have $sum(f(ts_i)) > sum(f(s_j))$ if and only if student i is a better tester than student j (similar for $sum(f(ts_i)) < sum(f(s_j))$). (Easy to prove). In case of equality between two students such that $sum(f(ts_i)) = sum(f(s_j))$:

For student i and j, get the array of the students that failed at the tests implemented by i and j respectively. So we'll have two arrays (obviously of the same size), with some students that can appear twice in the same array. From both array we'll compute a map reduce to get a map with as key the student and as value the number of test the student failed. Let call this map "relevant map". If the relevant map of i and j is the same, we can either choose that student i is better tester that student j or student j is better tester that student i: this will have no impact in the final grade (to be proved). If both relevant map are different, the instructor will have to choose either to give the point to the one that found a small number of "big" edge case that a lot of students failed on it (the relevant map with the higher number of students (key)), or to give the point to the one that found several "small" edge cases (the relevant map with the smallest number of students (key)). The instructor can also choose to let the random decide.

```
def partition(lambda_param, num_student):
  """
  lambda_param: the number must be between 0 and 1.
  num_student: the number of students that submitted test.
  """
  circular = 1
  partition_array = []
  for i in range(num_student - 1):
      new_lambda = 2 / (num_student - i) * lambda_param
      value = new_lambda * circular
      partition_array.append(value)
      circular = circular - value
  partition_array.append(circular)
  return partition_array
```

Algorithm 3: partition algorithm

Grading the tests - 4 The system wants to proceed as we did for the grade the solution side, but instead of increasing the grade for each passed test, we want to increase the grade for each function that failed on a given test. The initial grade here for the students can't begin at 0 since it's really hard to write test cases such that everyone failed at each test (i.e: the only way to get 100 if we proceed like we did for the solution side). Then, The instructor will have to input as parameter either the initial grade or the average he wants to have (see recalculate grades expected average algorithm). Then, we can proceed like we did (by two way) in the solution grade. Of course, instead of better tester we'll be interesting on the *better coder*.

Definition 3 (better coder) *The student that implements solution such that the sum of failed his solution has is the smallest. Pretty similar than better tester.*

Regarding the non perfect equality, the system choose at random.

3.6 Effort maximization

Definition 4 (Effort maximization) *A student reach an effort maximization if both axiom1 and 2 are respected.*

Theorem 2 (Effort maximization) *The P2PG system provide effort maximization for each arbitrary student.*

Proof 2 (Axiom 1) *In our system, only the test cases with at least one failure are consequent for the test grade, then, by assumption, the test case is not a wrong one, so everyone should pass it. Thus, the tests will remain not relevant for the grade computation. Here, we proved that the system provide a truthful tester.*

Proof 3 (Axiom 2) *Obviously, this is easy to detect, and trivially, the solution grade will be impacted by such a behavior, then there is no reason in the world to have such a behavior, and even if yes, the system already provide a penalty. Here, we proved that the system provide a truthful coder.*

Proof 4 (Effort maximization) *By proving the axiom 1 and 2, the effort maximization is proved.*

4 Smooth grades

4.1 The algorithm

At this stage, the grades are already computed. To get better results and to reinforce the relation between the tester side and coder side played by the same student, the instructor can choose to smooth the grades. This algorithm has been suggested by our advisor Dr Azaria Amos.

```
def smooth_grades(tests_dict, codes_dict,
    number_of_students, learning_rate):
    """
    tests_dict: python dict. The key is the name of the
        test, the value is a list
    with the first element being the weight and the rest
        being the user that failed at the test.
    codes_dict: The key is the name of the user, the
        value is a list
    with the first element being the weight and the rest
        being the tests that the user passed.
    number_of_students: Th number of student.
    learning_rate: 0.1 usually.
    """
    while(not tests_dict.converge() and not codes_dict.
        converge()):
         sum_of_all_weights = 0
         for test in tests_dict:
              value = sum_and_divide(number_of_students,
                  get_list(
                   codes_dict, tests_dict[test][1:]))
              tests_dict[test][0] = tests_dict[test][0] +
                  learning_rate * (value - tests_dict[test
                  ][0])
              sum_of_all_weights += tests_dict[test][0]
         for code in codes_dict:
              value = sum_weights_and_divide_weights(
                   sum_of_all_weights, get_list(tests_dict,
                       codes_dict[code][1:]))
              codes_dict[code][0] = codes_dict[code][0] +
                  learning_rate * (value - codes_dict[code
                  ][0])
```

Algorithm 4: smooth grades algorithm

Remark 2 (converge) *The smooth algorithm recalculate the weight of the test and of the code until convergence of both sides.*

Remark 3 (sum and divide) *Do sum of all the code weight and divide the result by the number of students.*

Remark 4 (sum weights and divide weights) *Do sum of the test weight passed by the coder and divide by the sum of all the tests weight.*

4.2 Example

Let's run an example. Before all things, let's define the table.

	u_1	u_2	u_3
t_1	FAILED	PASSED	PASSED
t_2	PASSED	PASSED	FAILED
t_3	PASSED	PASSED	FAILED
t_4	FAILED	FAILED	FAILED

Table 2: Example table

$\{u_1, u_2, u_3\}$ is the set of the student such that one student submit one solution. $\{t_1, t_2, t_3, t_4\}$ is the set of the UT submitted by students. We define the initial grades for the three students being: $g(u_1) = 50, g(u_2) = 75, g(u_3) = 25$. The initial weight for each test is 1.Let's now run the first round for the test 1 (t_1).

$w(t_1) = w(t_1) + \alpha * (w(t_1) - sum\_and\_divide(w(t_1))) = 1 + 0.1(\frac{50}{3} - 1) =$ 2.56666666667. In the second round the initial $w(t_1)$ value will be 2.6.

4.3 Efficiency

Here are some few theorems that the system wants to respect.

Theorem 3 *If the tests that user A passes is a subset of the tests passed by user B, so g(B)¿=g(A).*

Proof 5 *TODO.*

Theorem 4 *The algorithm always stop, that is the method always converges.*

Proof 6 *TODO.*

Theorem 5 *The grades are always converges between 0 and 1, and so are the tests weights.*

Proof 7 *TODO.*

5 Evaluation and results

6 Conclusions

The P2PG system presented in this paper is a new way to grades students. It includes two major improvements: a new method for grading assignment without bothering the instructor and it's force the student to familiarize with unit testing.

For the instructor side, in addition to not being bothering by grading assignment, it's a possibility for him to learn to the students how to implement UT. Of course, since our system wants to be as transparent as possible, he is still able to customize the grades by using parameter and choosing the best grading algorithm at his eyes.

For the student side, the process also include a re-read of the student with their code and others students codes. That is, combining all the phases, the student implements tests and code, he takes the roll of a grader and a graded, he is learning others students UT skills and can debate with others students about the UT they wrote. Of course, all the process gives him also a grade, for his coding skills and testing skill. That is, the student learned more than a simple assignment.

One of the biggest issue of the peer grading was the non-comprehension of the student regarding their grades. With the process presented in this paper, and since a feedback is constantly given to the student, this issue is solved. The method emphasizes that the student wants to progress and learn. Thus, even if it's remain easier to get a high grade, it's require a lot of work. And after all, the instructor can customize the grades.

7 Future work

The conflict phase can be see as useless. This phase allows to eliminates all the wrong tests from the system. We currently didn't found any replacement for this phase. It would be a great option to automatically remove every wrong tests without bothering students and instructor sides.

Our system could also combine the probabilistic methods used in the system of the Tuned Models of Peer Assessment in MOOCs [2] paper. That is, the system would compare the grade obtained by both method such that one method includes UT and the second not. It will allow to either recompute the grades of edit some few thing in the grade computation of the system.

There is an interesting learning work to do to always get better grades. Including deep learning or using probabilistic models.

References

[1] Boi Faltings Goran Radanovic and Radu Jurca. "Incentives for Effort in Crowdsourcing using the Peer Truth Serum". In: (2016).

[2] Chris Piech Jonathan Huang Zhenghao Chen Chuong Do Andrew Ng and Daphne Koller. "Tuned Models of Peer Assessment in MOOCs". In: (2013).