CrowdGrader: Crowdsourcing Homework Evaluation

Luca de Alfaro
UC Santa Cruz

Joint work with Michael Shavlovsky, UCSC
How can we crowdsource the evaluation of quality?
CrowdGrader lets students submit and collaboratively grade their solutions to homework assignments.

**STUDENTS**
Submit homework solutions and view prior submissions.
Review other student’s submissions, and manage your own reviews.

**TEACHERS**
View your assignments and create new ones
Manage your student lists.
1. Students submit their solutions (can submit in groups).

2. They are assigned submissions to review. They need to enter reviews, and grade/rank the solutions in order of quality.

3. Their final grade depends both on the quality of their submission, and on the quality of their reviewing work.
Q: How to assign reviews?

In conference reviewing, a set of items (papers) is assigned all at once.

Problems:

• **Missing reviews.** If some students don't do reviews, we may end up with some submissions with too few reviews.

• **Mix-ups.** Homework solutions are similar – students can mix up the homework and the reviews.

• **Suboptimal.** Premature allocation is suboptimal – we can always do better if we allocate as late as possible.
Q: How to assign reviews?

CrowdGrader assigns reviews **one at a time**.

- For each submission, it computes the number of likely reviews = done + recently assigned (uses some ML to predict).

- When a student finishes a review, it is assigned another submission to review chosen among those with least likely reviews.

Currently, this assignment is random, but there is much scope for improvement, matching via on-line algorithms the ability of the student, and the quality of the homework.

We are experimenting, but we don't have sufficient data yet.
Q: rank, or grades?

We initially asked students to rank the submissions they were reviewing in order of quality.

• Ranking requires only a relative, rather than an absolute judgement.

• We can then use both online, and offline, rank aggregation methods.
Combining rankings from different users

There are many ways.

Offline: rank aggregation

Borda's rule (1770):

- $n$ candidates.
- Each voter has gives $n$ points to top preference, $n-1$ for second preference, ..., 1 points to last preference.
- The candidate receiving the most points wins.

There has been extensive research up to the present day on rank aggregation methods.
This is a very good submission. The functionality is there, the code is well commented, and there is a lot of attention to the details of the UI. The only problem is that it’s not clear why some options are in the Settings menu, while others are in a setup screen; some settings reorganization would be advisable.
Ranking did not work well

<table>
<thead>
<tr>
<th>Assignment</th>
<th>$f_h$</th>
<th>Number of pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPS 121 hw 1</td>
<td>36%</td>
<td>252</td>
</tr>
<tr>
<td>CMPS 121 hw 2</td>
<td>41%</td>
<td>231</td>
</tr>
</tbody>
</table>

How often was the latest submission ranked above an already-reviewed one? (should be about 50%).

Ranking was often skipped:

• Uneasiness about ranking peers

• Considered a coarse instrument: students complained about having to arbitrarily rank submissions they considered "equivalent"

• Lack of trust that this would lead to an accurate ranking.
We moved to grades

- Initially, we forced students to assign a different (floating-point) grades to each submission.
  - Goal: force students to think about differences in quality.
  - Question: what's the probability that two different submissions have exactly the same quality?

- Students did not like this.

- Students like to put things into equivalence classes of quality.

- We gave in...
Optimal grade aggregation

<table>
<thead>
<tr>
<th>Users</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
</tr>
</tbody>
</table>

Average: 6.625
## Optimal grade aggregation

<table>
<thead>
<tr>
<th>Users</th>
<th>Grade</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>6.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>8.0</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Average: 6.625
Optimal grade aggregation

Minimum variance linear estimator:
Let $X_1, ..., X_n$ be uncorrelated random variables with mean $x$ and variances $\nu_1, ..., \nu_n$. The minimum-variance linear estimator of $x$ can be obtained by:

\[
\frac{\sum_{i=1}^{n} X_i / \nu_i}{\sum_{i=1}^{n} 1 / \nu_i}
\]

"Weigh $X_i$ by $1/\nu_i$"
Optimal grade aggregation

Minimum variance linear estimator:
Let \( X_1, \ldots, X_n \) be uncorrelated random variables with mean \( \mu \) and variances \( \sigma_1^2, \ldots, \sigma_n^2 \). The minimum-variance linear estimator of \( \mu \) can be obtained by:

\[
\frac{\sum_{i=1}^{n} X_i / \sigma_i}{\sum_{i=1}^{n} 1 / \sigma_i}
\]

Variance = \[\left( \sum_{i=1}^{n} \frac{1}{\sigma_i} \right)^{-1}\]
The Vancouver Algorithm

Iteratively:

• Use user variances to compute optimal estimates of item grades;
• Compare these optimal estimates for items with the grades given by the users, and obtain information on user variances.

The algorithm recalls maximum likelihood methods, and also [Iterative Learning for Reliable Crowdsourcing, Karger et al, 2011].
The Vancouver Algorithm

Build the bipartite graph \((I \cup U, E)\):

- \(I\): items
- \(U\): users
- \((i, u) \in E\) iff \(u\) reviewed \(i\).
The Vancouver Algorithm

Initialization step:
\[ \forall (i, u) \in E : \]

The user \( u \) sends to the item \( i \) the grade \( g_u(i) \), with variance \( v_{-i}(u) = 1 \).
Compute and propagate item estimates

\( \forall (i, u) \in E : \)

The item \( i \) considers the messages \( g_{u'}(i), v_{-i}(u') \) from users \( u' \neq u \).
The Vancouver Algorithm

Compute and propagate item estimates

\( \forall (i, u) \in E : \)

The item \( i \) considers the messages \( g_{u'}(i) \), \( v_{-i}(u') \) from users \( u' \neq u \).

Using \( g_{u'}(i) \) and \( v_{-i}(u') \),

it computes an estimated grade \( g_{-u}(i) \)

and an estimated variance \( v_{-u}(i) \),

and sends them to \( u \).
The Vancouver Algorithm

Compute and propagate user variances

∀ (i, u) ∈ E :
The user u considers the messages $g_{-u}(i')$, $v_{-u}(i')$ from items $i' \neq i$. 
The Vancouver Algorithm

Compute and propagate user variances

\[ \forall (i, u) \in E : \]

The user \( u \) considers the messages \( g_{-u}(i') \), \( v_{-u}(i') \) from items \( i' \neq i \). We let:

\[ v_{-i}(u) := \mathbb{E}[ (g_{u}(i') \! - \! g_{-u}(i'))^2 ] , \]

where:

- \( g_{u}(i') \) is the given by \( u \) to \( i \)
- \( g_{-u}(i') \) is the estimated grade
- every \( i' \) is given weight \( 1/v_{-u}(i') \)

We then send \( g_{u}(i) \), \( v_{-i}(u) \) to \( i \).
The Vancouver Algorithm

Final aggregation:
\[ \forall i \in I : \]

The item \( i \) considers all messages \( g_{u'}(i), v_{i}(u) \) from all connected \( u' \).

The item computes an optimal aggregation \( g(i) \).

\( g(i) \) is our “consensus grade”.
Vancouver: Performance on synthetic data

- 50 items, 50 users, 6 reviews per item.
- Users have gamma-distributed variance, with shape $k=2$, $k=3$.
- We report $\sigma = \sqrt{E[(\text{true} - \text{consensus})^2]}$
Vancouver: Performance on synthetic data

100 users
6 reviews per item
Review Incentive

- CrowdGrader computes a *crowd-grade* combining:
  - 75% : consensus grade of submission
  - 25% : review phase, composed of:
    - grade accuracy of user
    - helpfulness of reviews by student

The percentages can be varied by the instructor.
Grade Accuracy

Idea: compare to fully random grader.

\[ d(u) = \left( \sum_{i \in \partial u} (g_u(i) - g(i))^2 \right)^{\frac{1}{2}} \]

Average error of user u

\[ d_{\text{random}} = \left( \sum_{u \in U} \sum_{i \in \partial u} \sum_{j \in I} (g_u(i) - g(j))^2 \right)^{\frac{1}{2}} \]

Average error of fully random user (where the relation between grade and item is scrambled)

Grade accuracy: \[ a = 1 - \frac{\min(d(u), d_{\text{random}})}{2 \cdot d_{\text{random}}} \]

n: reviews done \hspace{1cm} N: reviews assigned
Review helpfulness

- Students can comment on the reviews they receive, and they can leave a feedback in \{-2, ..., 0, ..., +2\}. Based on this feedback, we compute the reviewer helpfulness:
  - We drop the lowest feedback, to avoid tit-for-tat, obtaining a list \( H \).
  - The helpfulness \( h \) is computed as follows (then clipped to \([0,1]\)):

\[
h = 0.7 \left( 1 + \frac{\sum_{x \in H} x \cdot w(x)}{2 \sum_{x \in H} w(x)} \right)
\]

where:

\[
w(x) = \begin{cases} 
1 & \text{if } x \geq 0; \\
2 & \text{if } x < 0.
\end{cases}
\]
Overall crowd-grade

The student crowd-grade $\gamma$ combines:

- Submission grade $g$
- Accuracy $a$
- Helpfulness $h$

\[
\gamma = (1 - \alpha)g + \alpha M \frac{(a + h) \min\{n, N\}}{2N}
\]

Where:

- $N = \text{num of reviews due}$, $n = \text{num of reviews done}$
- $[0, M] = \text{grading range}$
- $\alpha = 0.25$ (so, 75% of crowd-grade is due to $g$)
## Evaluation dataset

<table>
<thead>
<tr>
<th>Class</th>
<th>N students</th>
<th>N assign.</th>
<th>N req revs</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/Android</td>
<td>68</td>
<td>5</td>
<td>6</td>
<td>25%</td>
</tr>
<tr>
<td>CS/Web</td>
<td>78</td>
<td>2</td>
<td>5</td>
<td>25%</td>
</tr>
<tr>
<td>CS/C++</td>
<td>102</td>
<td>5</td>
<td>5</td>
<td>25%</td>
</tr>
<tr>
<td>CS/Java</td>
<td>22</td>
<td>1</td>
<td>4</td>
<td>25%</td>
</tr>
<tr>
<td>Eng/Essay1</td>
<td>55</td>
<td>2</td>
<td>5</td>
<td>2%</td>
</tr>
<tr>
<td>Eng/Essay2</td>
<td>232</td>
<td>1</td>
<td>6</td>
<td>15%</td>
</tr>
<tr>
<td>Econ</td>
<td>61</td>
<td>6</td>
<td>5</td>
<td>25%</td>
</tr>
</tbody>
</table>
## Participation in the review phase

<table>
<thead>
<tr>
<th>Class</th>
<th>N</th>
<th>Perc rev</th>
<th>Min rev</th>
<th>Avg rev</th>
<th>Avg len</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/Android</td>
<td>5</td>
<td>106%</td>
<td>50%</td>
<td>90%</td>
<td>203</td>
</tr>
<tr>
<td>CS/Web</td>
<td>2</td>
<td>95%</td>
<td>70%</td>
<td>96%</td>
<td>463</td>
</tr>
<tr>
<td>CS/C++</td>
<td>5</td>
<td>95%</td>
<td>56%</td>
<td>90%</td>
<td>406</td>
</tr>
<tr>
<td>CS/Java</td>
<td>1</td>
<td>100%</td>
<td>75%</td>
<td>105%</td>
<td>79</td>
</tr>
<tr>
<td>Eng/Essay1</td>
<td>2</td>
<td>78%</td>
<td>40%</td>
<td>77%</td>
<td>130</td>
</tr>
<tr>
<td>Eng/Essay2</td>
<td>1</td>
<td>92%</td>
<td>50%</td>
<td>89%</td>
<td>329</td>
</tr>
<tr>
<td>Econ</td>
<td>6</td>
<td>98%</td>
<td>82%</td>
<td>101%</td>
<td>163</td>
</tr>
</tbody>
</table>

- **N**: number of assignments
- **Perc rev**: percentage of students who reviewed
- **Min/avg rev**: minimum/average reviews by a submission, as %
- **Avg len**: average length of a review (characters)
Effect of on-line predictive review assignment

| Assignment | $|S|$ | RevsDue | MinRevs | AvgRevs |
|------------|-----|---------|---------|---------|
| CS/Android | hw 1 | 60      | 6       | 2       | 5.4     |
|            | hw 2 | 61      | 6       | 2       | 5.3     |
|            | hw 3 | 68      | 6       | 0       | 4.8     |
|            | hw 4 | 62      | 6       | 6       | 6.1     |
|            | hw 5 | 57      | 6       | 5       | 5.3     |
| CS/C++     | hw 1 | 102     | 5       | 0       | 4.6     |
|            | hw 2 | 97      | 5       | 3       | 4.6     |
|            | hw 3 | 91      | 5       | 4       | 5.1     |
|            | hw 4 | 97      | 5       | 3       | 4.6     |
|            | hw 5 | 90      | 5       | 4       | 5.1     |
## Variance of grades given to the same assignment

<table>
<thead>
<tr>
<th>Class</th>
<th>Average grade stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/Android</td>
<td>15.2%</td>
</tr>
<tr>
<td>CS/Web</td>
<td>10.4%</td>
</tr>
<tr>
<td>CS/C++</td>
<td>11.8%</td>
</tr>
<tr>
<td>CS/Java</td>
<td>14.5%</td>
</tr>
<tr>
<td>Eng/Essay1</td>
<td>8.0%</td>
</tr>
<tr>
<td>Eng/Essay2</td>
<td>8.2%</td>
</tr>
<tr>
<td>Econ</td>
<td>9.6%</td>
</tr>
</tbody>
</table>
Correlation between consensus grades and grades by TA / instructor

<table>
<thead>
<tr>
<th>Assignment</th>
<th>$\rho$</th>
<th>norm-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/C++ hw 2</td>
<td>0.75</td>
<td>14.0%</td>
</tr>
<tr>
<td>CS/C++ hw 3</td>
<td>0.84</td>
<td>14.9%</td>
</tr>
<tr>
<td>CS/Android hw 3</td>
<td>0.39</td>
<td>16.3%</td>
</tr>
<tr>
<td>CS/Java hw 2</td>
<td>0.85</td>
<td>17.5%</td>
</tr>
</tbody>
</table>

Problem: the TA grade can also be imprecise (TAs can fail to spot errors, just like students, etc).
## Difference in consensus grades received by pairs of identical submissions

<table>
<thead>
<tr>
<th>Assignment</th>
<th>D</th>
<th>N. pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/C++ hw 2</td>
<td>18.0%</td>
<td>6</td>
</tr>
<tr>
<td>CS/C++ hw 3</td>
<td>11.8%</td>
<td>12</td>
</tr>
<tr>
<td>CS/C++ hw 4</td>
<td>10.3%</td>
<td>20</td>
</tr>
<tr>
<td>CS/C++ hw 5</td>
<td>10.9%</td>
<td>20</td>
</tr>
</tbody>
</table>

\( D \) is the square root of the mean square difference of the grades received by identical submissions, expressed as a percentage of the maximum grade \( M \)
Review helpfulness

We have significative data for few assignments only.
Causes of error

Main causes of error for computer science:

• Students that cannot install/test software.
  – We had to let students *decline* reviews.

• Reviewers that miss features that are actually implemented:
  – Buttons too small, hidden in illogical places, etc.
  – Not really an error: UI matters also for homework

Very difficult to do something good on such random errors. TAs are not better.
What did the students think?

Liked:

• Able to look at other students' solutions.
  - Students who did not manage to get their code to work were able to run and examine several different working solutions.
  - Learn new ways of doing the work.
  - Realize how good others are.

• Amount and quality of feedback.
  - Much more detailed feedback than they got from a busy TA.

Disliked:

• Initially, distrustful of ranking w/o grades.
Thanks!

**CrowdGrader**: [www.crowdgrader.org](http://www.crowdgrader.org)

- Joint work with **Michael Shavlovsky**
- Built on **web2py**, with support from Massimo Di Pierro.
- Hosted on **Google Appengine** (full backups, redundant and scalable infrastructure).
- **Documentation** at [doc.crowdgrader.org](http://doc.crowdgrader.org)

CrowdGrader is a trade mark of CrowdGrader LLC.