UNIVERSITY OF CALIFORNIA
SANTA CRUZ

REPUTATION SYSTEMS AND INCENTIVES SCHEMES FOR QUALITY CONTROL IN CROWDSOURCING

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Michael B. Shavlovsky

June 2017

The Dissertation of Michael B. Shavlovsky
is approved:

Prof. Luca de Alfaro, Chair

Prof. John Musacchio

Dr. Atish Das Sarma

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Reputation Systems and Incentives Schemes for Quality Control in Crowdsourcing

by

Michael B. Shavlovsky

Crowdsourcing combines the abilities of computers and humans to solve tasks that computers find difficult. In crowdsourcing, computers process and aggregate input that is solicited from human workers; thus, the quality of workers' input is crucial to the success of crowdsourced solutions. Performing quality control at scale is a difficult problem: workers can make mistakes, and computers alone, without human input, cannot be used to verify the solutions. We develop reputation systems and incentive schemes for quality control in the context of different crowdsourcing applications.

To have a concrete source of crowdsourced data, we built CrowdGrader, a web based peer grading tool that lets students submit and grade solutions for homework assignments. In CrowdGrader, each submission receives several student-assigned grades which are aggregated into the final grade using a novel algorithm based on a reputation system. We first overview our work and the results on peer grading obtained via Crowdgrader. Then, motivated by our experience, we propose hierarchical incentive schemes that are truthful and cheap. The incentives are truthful as the optimal worker behavior consists in providing accurate evaluations. The incentives are cheap as they leverage hierarchy so that they be effected with a small amount of supervised evaluations, and the strength of the incentive does not weaken with increasing hierarchy depth. We show that the proposed hierarchical schemes are robust: they provide in-

centives in heterogeneous environments where workers can have limited proficiencies, as long as there are enough proficient workers in the crowd. Interestingly, we also show that for these schemes to work, the only requisite is that workers know their place in the hierarchy in advance.

As part of our study of user work in crowdsourcing and collaborative environments, we also study the problem of authorship attribution in revisioned content such as Wikipedia, where virtually anyone can edit an article. Information about the origin of a contribution is important for building a reputation system as it can be used for assigning reputation to editors according the quality of their contribution. Since anyone can edit an article, to attribute a new revision, a robust method has to analyze all previous revisions of the article. We describe a novel authorship attribution algorithm that can scale to very large repositories of revisioned content, as we show via experimental data over the English Wikipedia.

To Katherine, for her support

Acknowledgments

I am grateful to Luca for his guidance and support. Working with him side by side was one of the most enriching and fun experiences during the years. I want to thank the memebers of the committe, Atish and John, for their discussions and insightful feedback. I also learned a great deal from my co-authors, labmates and friends. Thanks to Vassilis, Daniel, Zhongpeng, Marco, Huascar, and Maria. Last but not least, I am indebted to my family who keep believing in me during the hardest times. Thanks to Kat, Mom, Dad, and the rest of the family, for everything you did for me. It would have been impossible without you.

Chapter 1

Introduction

1.1 Quality Control in Crowdsourcing

Computers enable humans to solve tasks that could not be solved by humans alone. Computers can easily process large amounts of data, perform complex arithmetic computations, and so forth. However, there are tasks that are hard or impossible to solve using computers alone, without any human input. People can answer questions that are unique to human experience. They can transfer knowledge and concepts from one domain to another. As a result, humans can easily judge the quality of a movie, write a description for a product, or describe a meaning of a sentence.

Human computation and crowdsourcing [57] is a method of solving tasks by outsourcing them to a large group of workers. Workers are usually distributed and submit solutions via the Internet. Crowdsourcing can offer improvements in cost, speed, scalability and diversity. Researchers, universities and companies use crowdsourcing to solve a wide range of tasks: data

annotation, grading of homework submissions, writing product descriptions, surveys, and so forth.

There are many examples of problems that have been solved using crowdsourcing. Galaxy Zoo is a crowdsourcing project with the goal of classifying galaxies in telescope images. Hundreds of thousands of volunteers helped classify millions of galaxies. The unprecedented scale allowed researchers to classify a large number of galaxies much faster, compared to the original approach when only astronomy researchers and graduate students classified galaxies. Universities and Massive Open Online Courses (MOOCs) use crowdsourcing in classes to evaluate homework submissions. Peergrading is the practice when students grade each others homeworks submissions; students serve both roles as the providers of tasks and as the workers. Peergrading is especially popular in large classes as it helps reduce the grading burden on instructors and teaching assistants. Companies use crowdsourcing to solve business related tasks such as writing product descriptions, surveys, annotating datasets for machine learning algorithms, transcribing scanned receipts, and others. Another prominent example is Wikipedia, The Free Encyclopedia. Editors from all over the world write articles on a wide range of topics, from classic articles one can find in an encyclopedia, to articles covering recent news.

Quality control is a central problem in crowdsourcing. The very nature of crowdsourcing makes quality control a challenging problem. Workers can be unreliable, and computers alone, without human input, cannot be used to evaluate workers. A natural solution is to employ experts that evaluate the workers. However, experts are usually expensive, and the total price of quality control at scale can be too large. In the extreme case, having experts evaluate all workers is equivalent to the experts doing all the work.

Reputation systems and incentives schemes are tools for improving the quality of contributions obtained via crowdsourcing. Reputation systems [71] measure reliability of workers based on available signals. For example, if workers have common tasks then reputation can be estimated via a comparison of answers on those tasks. Peer prediction [58] is an approach of estimating the quality of answers by measuring agreement between peers. However, reputation systems based on peer prediction are prone to collusions. As an illustration, if all workers agree to report the same answer on any tasks, then workers appear to be in perfect agreement with each other, while providing no useful information.

Mechanism design is a subfield of economics [27] that studies incentives schemes in settings where players (workers) act strategically. A successful incentive scheme takes into account workers' strategic choices and guarantees that a diligent work is the optimal worker behavior.

In this work we developed reputation systems and incentives schemes for quality control in crowdsourcing.

1.2 Outline of the Dissertation

To investigate the algorithms and incentives that can be used in crowdsourcing quality evaluations, we built CrowdGrader, a tool that lets students submit and collaboratively grade solutions to homework assignments. In Chapter 2, we present the algorithms and techniques used in CrowdGrader, and we describe our results and experience in using the tool for several computer-science assignments. CrowdGrader combines the student-provided grades into

a consensus grade for each submission using a novel crowdsourcing algorithm that relies on a reputation system. The algorithm iteratively refines inter-dependent estimates of the consensus grades, and of the grading accuracy of each student. On synthetic data, the algorithm performs better than alternatives not based on reputation. On our experimental data, the performance seems dependent on the nature of review errors, with errors that can be ascribed to the reviewer being more tractable than those arising from random external events. To provide an incentive for reviewers, the grade each student receives in an assignment is a combination of the consensus grade received by their submissions, and of a reviewing grade capturing their reviewing effort and accuracy. This incentive worked well in practice.

We also study the factors that influence errors in peer grading. We analyze 288 assignments with 25,633 submissions and 113,169 reviews conducted with CrowdGrader. First, we found that large grading errors are generally more closely correlated with hard-to-grade submissions, rather than with imprecise students. Second, we detected a weak correlation between review accuracy and student proficiency, as measured by the quality of the student's own work. Third, we found little correlation between review accuracy and the time it took to perform the review, or how late in the review period the review was performed. Finally, we found a clear evidence of tit-for-tat behavior when students give feedback on the reviews they received. We conclude with remarks on how these data can lead to improvements in peer-grading tools.

In Chapter 3, we develop incentives schemes for crowdsourcing problems where the workers are asked to provide evaluations for items; the worker evaluations are then used to estimate the true quality of items. Lacking an incentive scheme, workers have no motive for making an effort in completing the evaluations, providing inaccurate answers instead. We show that a

simple approach of providing incentives by assessing randomly chosen workers is not scalable: to guarantee an incentive to be truthful the number of workers that the supervisor needs to assess grows linearly with total number of workers. To address the scalability problem, we propose incentive schemes that are truthful and cheap: the truthful as the optimal worker behavior consists in providing accurate evaluations, and cheap because the truthfulness is achieved with little supervision cost. We consider both discrete evaluation tasks, where an evaluation can be done either correctly, or incorrectly, with no degrees of approximation in between, and quantitative evaluation tasks, where evaluations are real numbers, and the error is measured as distance from the correct value. For both types of tasks, we develop hierarchical incentive schemes that can be effected with a small amount of supervised evaluations, and that scale to arbitrarily large crowd sizes: they have the property that the strength of the incentive does not weaken with increasing hierarchy depth. We show that the proposed hierarchical schemes are robust: they provide incentives in heterogeneous environments where workers can have limited proficiencies, as long as there are enough proficient workers in the crowd. Interestingly, we also show that for these schemes to work, the only requisite is that workers know their place in the hierarchy in advance.

Determining who is the author of a piece of text is an important problem in crowdsourced and revisioned content such as Wikipedia. The knowledge on authorship can help determine user reputation [1]. In Chapter 4 we consider the problem of attributing authorship to such revisioned content, and we develop scalable attribution algorithms that can be applied to very large bodies of revisioned content, such as the English Wikipedia. Since content can be deleted, only to be later re-inserted, we introduce a notion of authorship that requires comparing each new revision with the entire set of past revisions. For each portion of content in the

newest revision, we search the entire history for content matches that are statistically unlikely to occur spontaneously, thus denoting common origin. We use these matches to compute the earliest possible attribution of each word (or each token) of the new content. We show that this "earliest plausible attribution" can be computed efficiently via compact summaries of the past revision history. This leads to an algorithm that runs in time proportional to the sum of the size of the most recent revision, and the total amount of *change* (edit work) in the revision history. This amount of change is typically much smaller than the total size of all past revisions. The resulting algorithm can scale to very large repositories of revisioned content, as we show via experimental data over the English Wikipedia.

Chapter 2

Crowdsourcing Quality Evaluations in the Context of Peergrading

2.1 Problem Setting and Contributions

Ranking items according to their quality is a universal problem, occurring when hiring, admitting students, accepting conference papers, presenting search results, selecting winners in contests, and more. Often, the quality of items is best judged by human evaluators. As relying on a single evaluator is often impractical — and can be perceived as unfair — an overall evaluation can be obtained via crowdsourcing: several evaluators compare or grade a subset of the items, and their feedback is then combined in an overall ranking or scoring of the items.

To study the algorithms and incentives that can be used in crowdsourcing quality evaluations, we built CrowdGrader, a tool for the crowdsourced evaluation of homework assignments. CrowdGrader lets students submit, and collaboratively grade, solutions to home-

work assignments; their grade for each assignment depends both on the quality of their submitted solution, and on the quality of their work as graders. CrowdGrader is available at `http://www.crowdgrader.org/`.

We chose to focus on homework grading for several reasons. First, this is a problem that we know well, and we were confident that the tool would be used by us and by some of our colleagues, providing valuable experimental data. Furthermore, solutions submitted to a homework assignment share the same topic: we do not need to address the problem of matching the topic of each submission to the domain of expertise of each reviewer, as it is necessary for conference submissions. The students submitting the homework solutions would provide a ready pool of graders. Last, but not least, we hoped the tool would provide educational benefits to the students. We hoped students would benefit from being able to examine the solutions submitted by other students: accomplished students would be able to look at alternative ways of solving the same problem, and students who encountered difficulties would be able to study several working solutions to the problem while grading. We also hoped that students would benefit from their peer's feedback.

The first question we studied with the help of CrowdGrader concerned the algorithms that can be used to merge the grades provided by each evaluator, into overall *consensus* grades for each assignment. We developed a novel crowdsourcing algorithm, which we nicknamed `vancouver`, that combines the grades provided by the students with the help of a reputation system that captures the student's grading accuracy. The algorithm proceeds via iterations, following a structure inspired by [48], and inspired also by expectation maximization techniques [24, 21, 70]. In each iteration, the algorithm computes a consensus estimate of the grade of each

submission, weighing the student input according to the accuracy of each student; the consensus estimates are then used to update the estimated accuracy of the students. On synthetic data, `vancouver` performs well, far outperforming algorithms such as the average or median. On our real-world data, the results are mixed. Our impression is that `vancouver` outperforms simpler algorithms when the grading errors of the students are not random. However, in one of the classes where CrowdGrader was used, the grading errors were random in nature, due to mismatches between the code compilation environments of the students submitting and evaluating homework solutions, and in that case, `vancouver` performed slightly worse than simple average.

The second question we studied concerned the incentives necessary to obtain quality evaluations from the students. Our approach was simple: we made the grade each student received depend on both the quality of the solution they submitted, and on the quality of their review and grading work. This worked well in practice, and we will describe the methods we used for assigning grading credit.

The third question we studied concerned the factors cause or influence the errors in peer-assigned grades? We analyze 288 assignments with 25,633 submissions and 113,169 reviews conducted with CrowdGrader. First, we found that large grading errors are generally more closely correlated with hard-to-grade submission, rather than with imprecise students. Second, we detected a weak correlation between review accuracy and student proficiency, as measured by the quality of the student's own work. Third, we found little correlation between review accuracy and the time it took to perform the review, or how late in the review period the review was performed. Finally, we found a clear evidence of tit-for-tat behavior when students give feedback on the reviews they received. We conclude with remarks on how these data can

lead to improvements in peer-grading tools.

2.2 Related Work

The work most closely related in goals to ours is the proposal to crowdsource the review of proposals for use of telescope time by [56], as well as the recent NSF pilot project for reviewing funding proposals [31]. As in those approaches, we also distribute the task of reviewing the submissions to the same set of people who submitted the items to be reviewed. Both for proposals submitted to a specific panel, and for solutions submitted to the same homework assignment, the submissions are on sufficiently related topics that the problem of matching submission topic with reviewer expertise can be disregarded. For proposals, of course, care must be taken to avoid conflicts of interest; our situation for homeworks is relatively simpler. Where the problems differ is that proposal reviewing is essentially a top-k problem: the best k proposals must be selected for funding. Homework grading, on the other hand, is an evaluation problem: each item needs to be graded on a scale. In top-k problems, the most important consideration is precision at the top; mis-ranking items that are far from the top-k carries no real consequence. In our evaluation problem, each evaluation carries approximately the same importance, and we do not need to precisely rank students whose submissions have approximately the same quality. While there are techniques that can be applied to both problems, this difference in goals justifies the reliance of [56, 31] on comparisons, and ours on grades. Comparisons can allow the precise determination of the top-k items in a ranking [19]; we chose instead to develop reputation-based algorithms for merging grades.

The works of [56] and [31] discuss incentive mechanisms for reviewers, consisting in awarding a better placement in the final ranking to proposals whose authors did a better job of reviewing. We follow the same approach, but we have the additional constraint that students must find the reviewing work appropriately rewarded with respect to the time it takes. Students are most often under time pressure, and they often consider the question of whether one hour is better spent reviewing for one class, or working on the homework assignment of another. A reward such as the one of [31], where a couple of places in the ranking are awarded based on reviewing work, would not have sufficed, especially in the context of an evaluation rather than top-k setting. Rather, we let instructors chose a reward magnitude that is commensurate with the time required by reviewing. Furthermore, unlike [56, 31], we face the additional constraint that students must regard the reward as fair and non-punitive; as we will see in Section 2.6, this affected our choice of reward metrics.

The effect of review incentive on the quality of the ranking is examined in depth in [60]. The main problem, also raised in [56], is that the incentive mechanism makes the grading a "Keynesian beauty contest", where reviewers are rewarded for thinking like other reviewers; in turn, this may encourage a "race to mediocrity", in which non-controversial, blander proposals may fare better than more audacious and original ones. We agree with the authors of [60] that this may be a true problem for proposal review. However, we believe that in the context of homework assignments, the problem may be minor or non-existent. Our incentive function, described in Section 2.6, gives a farily generous reward that would not overly decrease if a students mis-ranks one of the assignments; this gives more leeway to students presented with a homework submission that does not follow the beaten path. We also believe that the less

competitive evaluation setting, as compared to a top-k setting, may lessen the problem. Finally, in our somewhat limited real-world experience, students generally were more ready to reward originality than teaching assistants. The main goal of a teaching assistant is often to avoid controversy, in order to avoid confrontations with students. Thus, teaching assistants generally felt a stronger obligation to follow a rigid grading scheme, for the sake of consistency, and subtract a fixed number of points for each type of error encountered. Students felt less constrained by the need for full consistency, as the authors of the submissions they graded could not easily identify or compare the grades they received from the same grader.

The reputation-based crowdsourcing algorithm we use to aggregate grades is inspired by the algorithm of [48] for the aggregation of boolean input. Unlike that work, however, we do not have a proof of convergence for our crowdsourcing algorithm, nor a full theoretical characterization of how the precision depends asymptotically on the number of reviews. The algorithm is also inspired, and related, to the technique of expectation maximization [24, 21, 75, 43, 83, 70, 82]. The approach is also related to belief propagation methods [63, 89]. A related, but coarser, method was used by one of the authors to aggregate information provided by editors of Google Maps via the Crowdsensus system [22].

Rank aggregation methods have a very long history. The problem originally arose in the context of elections. In a classical contribution [23], de Borda proposed that each voter assigns each of n candidates a score $1, 2, \ldots, n$, according to the preference; the candidates were then ranked according to the total score they received from all voters. Again in the context of elections, Arrow proved a famous theorem, stating that any rank aggregation that satisfies transitivity, unanimity, and independence of irrelevant alternatives is a dictatorship, where there

is a single fixed voter (the dictator) who determines the outcome [6, 34]. An overview of rank aggregation methods used in democracies around the world can be found in [53].

Kemeny-optimal rankings minimize the sum of Kendall-Tau distances between the ranks proposed by individual voters, and the aggregate rank. The problem of computing Kemeny-optimal rankings is known to be NP-hard [7, 25]. Cynthia Dwork, Ravi Kumar et al. [25] study approximation methods that can be applied to the problem of ranking search results by combining the output of several rankers. Nir Ailon et al. [4] developed an algorithm to find approximate solution subject to additional constraints. The problem of finding Kemeny optimal solution is equivalent to the minimum feedback arc set problem, and Kenyon-Mathieu and Schudy [50] obtained polynomial time algorithm for computing a solution with loss at most $(1+\varepsilon)$.

On-line algorithms for rank aggregation have been long studied, especially in their application to ranking in sports such as chess and tennis. In these algorithms, a global ranking is gradually refined and updated according to a stream of incoming comparisons. In sports, these comparisons consist in the outcomes of matches between players; in other settings, the comparisons may be obtained by asking users or visitors to sites to select a winner among a set of alternatives. In the original paper by Bradley and Terry, the player strenghts are obtained from match outcomes via a maximum-likelyhood approach [9, 54]; Elo replaced this with a dynamic update process which could account also for the time-varying aspect of player strenghts [26]. Glickman then refined the models and the algorithms by first adapting a Bayesian update approach [38], and by then obtaining efficient algorithms via approximation and parameter estimation [37].

The accuracy of peer grading in the context of MOOCs has been analyzed in [52],

where the match between instructor grade and student grades is analyzed in detail. The study finds a tendency by student to rate higher people that share their country of origin — and this in spite of the grading process being anonymous. The study finds that improvement in grading rubrics lead to improved grading accuracy. Geographical origin, along with gender, employment status, and other factors, are found to have influence on engagement in peer grading in a French MOOC in [16]. Our work is thus somewhat orthogonal to [16, 52]: we do not have data on student ethnicity, and we focus instead on factors measurable from the peer grading activity itself.

Frequently, peer grades are accompanied with reviewers' comments or feedback; [88] explores the possibility of using the review text to asses review quality. The authors show a successful application of classifiers and statistical Natural Language Processing to evaluate reviews.

Peer Instruction is a process in which students can observe grades by other reviewers, discuss the review, and consequently modify their grades [18]. The factors that influence grades in peer instruction have been studied in [8]. In spite of the different settings, [8] also observe that the behavior of high and low-scoring students is fairly similar in terms of their grading accuracy.

2.3 CrowdGrader

CrowdGrader lets students submit and collaboratively grade solutions to homework assignments. The lifecycle of an assignment in CrowdGrader consists of three phases: a sub-

mission phase, a review phase, and a grading phase.

The submission phase is standard.

In the review phase, each student must review a given number of submissions. The more submissions each student reviews, the more accurate the crowd-sourced grade will be, but the larger the workload on the students. In our experiments, asking that each submission was reviewed by 5 or more students yielded acceptable accuracy.

Once the review period is over, CrowdGrader computes a consensus grade for each submission, aggregating the grades or comparisons provided by the students via the algorithms we will present in Section 2.5. Crowdgrader then assigns a *"crowd-grade"* to each student, by combining the consensus grade of the submission with a *review grade* which quantifies the review effort and accuracy of each student. In our experiments, computing the crowd-grade by giving 75% weight to the submission grade, and 25% to the review grade, provided sufficient motivation for the students to put adequate effort in reviewing. The instructors can either use the crowd-grade as the grade for the student in the assignment, or they can fine-tune the final grades, for instance to correct overall biases.

We applied CrowdGrader to the grading of coding assignments, namely, Android programming assignments (CMPS 121, taught by one of the authors at UCSC); C++ programming assignments (CMPS 109, also taught at UCSC); and Java assignments (taught at University of Naples). While CrowdGrader can support in principle many types of assignments, we focused on programming assignments for three reasons.

Programming assignments are especially burdensome to grade: unpacking, compiling, and testing each submission is a time-consuming process. CrowdGrader enabled us to give

coding assignments weekly, spreading what would have been a very onerous grading task on the students participating in the class.

Second, we thought that students would be able to test and evaluate the submitted code with reasonable accuracy.

Third, we believed that students would directly benefit from reading the code submitted by other students. Strong students would be presented with alternative ways of solving the problems, and weaker students would have an opportunity to study several working solutions. Indeed, students reported a positive experience from the tool, citing their ability to learn from others, and at the usefulness of the feedback they received, as the main benefits.

The code for CrowdGrader as used for this paper is available from `https://github.com/lucadealfaro/crowdranker`, and CrowdGrader itself is available at `http://www.crowdgrader.org/`.

2.4 Design of the Review Phase

The review phase is of primary importance for the accuracy of the generated ranking, and we experimented with several designs.

2.4.1 Review assignment

We opted for an anonymous review process, in which submissions to review were assigned automatically to students. Since students could not choose which submissions to review, nor in general did they know the identity of the submissions' authors, they had limited ability

to collude and cause their friends to receive higher grades.

In usual computer-science conferences, papers are assigned to program-committee members in a single batch; each member then has a period of time to read the papers and enter all reviews. We decided to follow a different approach, in which submissions were assigned to students for review one at a time: students were assigned a new review task only upon completion of the previous one. Our chief concern in making this decision was to ensure that students would not get the submissions, and their reviews, mixed up. Unlike conference papers, the submitted homework solutions are all on the same topic, and they can be fairly similar to each other; furthermore, to preserve anonymity, submissions under review were denoted by un-memorable names such as "Homework 2 Assignment 3". By having students work on one review at a time, we hoped to cut down on the possibility of mix-ups. Indeed, we received no valid reports of mis-directed reviews.

Delaying the review assignment until the last moment offered two additional benefits. First, we were able to ensure that all submissions received roughly the same number of reviews, even if some students failed to do any reviewing work. For each submission, we considered the number of *likely reviews,* consisting of the completed reviews, along with the review tasks that had been assigned only a short time before. When assigning reviews, we chose submissions having least number of likely reviews. Second, the delayed assignment let us gather information about the quality of submissions, as the review process proceeded, enabling us to optimize the review assignment by routing submissions to students who were in the best position to provide feedback on them. We have experimented with various techniques for routing submissions, but we do not yet have sufficient experimental evidence to report on the performance of the

algorithms.

2.4.2 Comparisons vs. grades

For the first homework assignment conducted using CrowdGrader, we decided to ask students to rank homework submissions, rather than grade them. We had more faith in the students' ability to compare submissions, than in their ability to assign grades with sufficient consistency, so that grades assigned by different students would be comparable. When reviewing a submission, students were presented with a screen displaying the submissions they had already ranked, in the quality order they had previously entered; at the bottom, and in a highlighted color, was the new submission to review. Students were instructed to write some feedback for the submission's author, and then to drag and drop the new submission into the appropriate place in the ranking.[1]

Unfortunately, after writing the feedback paragraph, many students skipped the ranking step, leaving the new submission where they found it — at the bottom of the ranking. To confirm this, we measured the fraction of times f_h that students would rank the newly assigned submission higher than a given submission they had already reviewed. Had students been accurate, this fraction should have been close to 50%, since there was no relationship between the quality of the new submissions, and that of the previously-reviewed ones. Instead, in the first assignment this fraction was only 36%. Even after strongly reminding students to provide a ranking, the fraction f_h rose only to 41% in the second assignment. Table 2.1 reports the value of f_h for the five CMPS 121 Android assignments.

[1] While inserting the new submission in the ranking, the students were able to re-order previously ranked submissions.

Assignment	f_h	Number of pairs
CMPS 121 hw 1	36%	252
CMPS 121 hw 2	41%	231
CMPS 121 hw 3	53%	271
CMPS 121 hw 4	52%	277
CMPS 121 hw 5	49%	221

Table 2.1: Fraction f_h of pairs consisting of a previously-reviewed submission, and a submission under review, in which the submission under review was ranked higher by the student than the previously-reviewed one.

Talking to students, we understood that they were skipping the ranking step because of a combination of forgetfulness, and unwillingness. Several students mentioned that they felt unconfortable with providing a ranking of their peers. Furthermore, they thought that ranking was a blunt instrument. They complained about having to arbitrarily rank submissions that they felt were roughly equivalent, and they worried that ranking did not differentiate between the situations of submissions of roughly equivalent quality, and submissions of widely different quality. While ranking can indeed be precise, we are concerned not only with precision, but also with how the tool is received by the students.

The problem in our UI, of course, was that we could not distinguish between a skipped ranking, and a valid ranking. Starting from the third homework assignment, we modified the UI so that students needed to *both* rank the submissions, *and* assign a grade to each one: the ranking had to reflect the grades. As students could not leave grades blank, this effectively forced students to provide a valid ranking. Table 2.1 shows that from assignment 3 onwards the fraction f_h was very close to 50%. Adding grades led to a more accurate ranking of the submissions — regardless of whether the grades themselves were used! The student satisfaction

with CrowdGrader also markedly increased, once grades were seen as the primary method of providing input to the tool.

Once grades were available, we decided to use the additional information they convey, and we focused on the development of crowdsourcing algorithms for the aggregation of grades. In the current UI of CrowdGrader, students still need to both rank submissions, and assign them a grade. Obviously, once we have grades, the ranking step is un-necessary. However, we believe that asking students to also rank the submissions forces them to consider the relationship between submissions with similar grades, leading them to fine-tune the grades to more accurately reflect their quality assessment. We intend to confirm this belief in future work, comparing the accuracy of the grades with, and without, the ranking step.

2.4.3 Rejecting evaluations

We discovered early on that it was important to allow students to leave some submissions ungraded, and yet consider their reviewing duty for the submission as completed, as far as the computation of the students' own review grades were concerned. In our programming assignments, there were many cases in which well-intentioned students were unable to review submissions. In the Android class, their installation of Eclipse and Android SDK occasionally misbehaved in a way that left students unable to load and review the code submitted by other students. In the C++ class, glitches or differences in the build environment occasionally prevented students from compiling and executing the submissions under review. Initially, students needed to enter a grade to receive credit for their review effort, and students entered very low grades for the submissions they could not evaluate. In our informal analysis of the accuracy of the tool,

this was the largest source of discrepancy in the grades assigned by different students to the same submission. The solution was to let students flag a review task as "declined", omitting the grade, and providing instead an explanation of why they were declining it. In our experiments, no more than 1% of submissions required instructor evaluation, since all students declined their review; these submissions typically were markedly incomplete and non-functional.

2.5 The Vancouver Crowdsourcing Algorithm

Once students assign grades to the submissions they review, we need to aggregate the student-provided grades into a *consensus* grade for each submission. The simplest algorithm for computing consensus grades consists in averaging the grades each submission has received; we refer to this algorithm as `avg`. We developed an alternative algorithm, the `vancouver` algorithm.[2] The `vancouver` algorithm measures each student's grading accuracy, by comparing the grades assigned by the student with the grades compared to the same submissions by other students, and gives more weight to the input of students with higher measured accuracy. The algorithm thus implements a reputation system for students, where higher accuracy leads to higher reputation, and to higher influence on the consensus grades.

On synthetic data, `vancouver` is far more accurate than `avg`. On our experimental data, `vancouver` performs better than `avg`, but as we will report in Section 4.6, the difference is not quite as large, perhaps due to the fact that our assumptions about user behavior do not fully correspond to how students behave in practice.

[2]The algorithm owes its name to the fact that it was conceived while strolling the pleasant streets of this Canadian city.

2.5.1 Variance minimization principle

The `vancouver` algorithm is based on the following fact.

Proposition 1. (minimum variance estimator) *Suppose we have available uncorrelated estimates $\hat{X}_1, \ldots, \hat{X}_n$ of a quantify x of interest, where each $\hat{X}_i$ is a random variable with average x and variance v_i, for $1 \leq i \leq n$. We can obtain an estimate of x that has minimum variance by averaging $\hat{X}_1, \ldots, \hat{X}_n$ while giving each $\hat{X}_i$ a weight proportional to $1/v_i$, for $1 \leq i \leq n$. That is, the minimum variance estimator $\hat{X}$ of x can be obtained as:*

$$\hat{X} = \frac{\sum_{i=1}^{n} \hat{X}_i / v_i}{\sum_{i=1}^{n} 1/v_i} \quad .$$

The variance of this estimator is

$$\mathrm{var}(\hat{X}) = \left(\sum_{i=1}^{n} \frac{1}{v_i} \right)^{-1} \quad .$$

Proof. Given two uncorrelated estimates $\hat{X}_1, \hat{X}_2$, with variances v_1, v_2, consider their linear combination $Y = \alpha_1 \hat{X}_1 + \alpha_2 \hat{X}_2$, with $\alpha_1 + \alpha_2 = 1$. By the Bienaymé formula, the variance of Y is given by $\alpha_1^2 v_1 + (1-\alpha_1)^2 v_2$.. If we take the derivative with respect to α_1, and set it to 0, we obtain $\alpha_1 v_1 = \alpha_2 v_2$, or $\alpha_1 \propto 1/v_1$ and $\alpha_2 \propto 1/v_2$. The general case for n estimates follows similarly. ∎

This observation immediately suggests how to obtain reputation-based crowdsourcing algorithms for grades: if we could somehow measure the variance v_i of each student i, we could weigh the input provided by student i in proportion to $1/v_i$.

2.5.2 Algorithm structure

We developed an algorithm that proceeds in iterative fashion, using consensus grades to estimate the grading variance of each user, and using the information on user variance to compute more precise consensus grades. The structure of the algorithm is inspired by the algorithm of [48] for computing consensus boolean values. To state the algorithm, we denote by U the set of students, and by S the set of items to be graded (the submissions). We let $G = (T, E)$ be the graph encoding the review relation, where $T = S \cup U$ and $S \cap U = \emptyset$, and where $(i, j) \in E$ iff j reviewed i; for $(i, j) \in E$, we let g_{ij} be the grade assigned by j to i. We denote by ∂t the 1-neighborhood of a node $t \in T$.

The algorithm proceeds by updating estimates v_j of the variance of user $j \in U$, and estimates c_i of the consensus grade of item $i \in S$, and estimates v_i of the variance with which c_i is known. To produce these estimates, the algorithm relies on messages $m = (l, x, v)$ consisting of a source $l \in S \cup U$, of a value x, and of a variance v. We denote by M_i, M_j the lists of messages associated with item $i \in T$ or user $j \in U$. Given a set M of messages, we indicate by

$$E(M) = \frac{\sum_{(l,x,v)\in M} x/v}{\sum_{(l,x,v)\in M} 1/v}$$
$$\mathrm{var}(M) = \left(\sum_{(l,x,v)\in M} \frac{1}{v} \right)^{-1}$$

the best estimator we can obtain from M, and its variance.

The details are given in Algorithm . Lines 2–4 initialize the messages to items using the grade assigned by the users, and a constant variance (whose precise value is unimportant). If

Algorithm The Vancouver Algorithm.

Input: A review graph $G = ((S \cup U), E)$ such that $|\partial t| > 1$ for all $t \in S \cup U$, along with $\{g_{ij}\}_{(i,j)\in E}$, and number of iterations $K > 0$.

Output: Estimates $\hat{q}_i$ for $i \in S$.

{Initialization}
for all $i \in S$ **do**
  $M_i := \{(j, g_{ij}, 1) \mid (i,j) \in E\}$.
end for
for iteration $k = 1, 2, \ldots, K$ **do**
  {Propagation from items}
  for all $j \in U$ **do**
    $M_j := \emptyset$
  end for
  for all $i \in S$ **do**
    for all $j \in \partial i$ **do**
      Let $M_{-j} = \{(j', x, v) \in M_i \mid j' \neq j\}$ in $M_j := M_j \cup (i, E(M_{-j}), \mathrm{var}(M_{-j}))$
    end for
  end for
  {Propagation from users}
  for all $i \in S$ **do**
    $M_i := \emptyset$
  end for
  for all $j \in U$ **do**
    for all $i \in \partial j$ **do**
      Let $M_{-i} = \{(i', (x - g_{i'j})^2, v) \mid (i', x, v) \in M_j, i' \neq i\}$ in $M_i := M_i \cup (j, g_{ij}, E(M_{-j}))$
    end for
  end for
end for
{Final Aggregation}
for all $i \in S$ **do**
  $\hat{q}_i := E(M_i)$
end for

we had a-priori information on the variance of some users, it could be used in this initialization step. Lines 7–14 propagate, from items to the users who graded them, the best estimate available on the item grades and variances. In line 12, when we compute the estimate that is sent to each user, we do not use information coming from that same user. Lines 16–23 propagate, from users to the items they graded, the (immutable) grade the user assigned to the item, and a newly-recomputed estimate of the user's grading variance. The estimate of the user variance is computed by considering the differences between the item grades assigned by the user, and the estimates received from the items. Again, when computing the user variance that will be sent to an item, we do not consider the contribution to the variance due to this same item. Finally, in lines 26–28 we aggregate the information from users into our final estimates of item grades. We note that we gave above the most concise presentation of the algorithm; a more efficient implementation can be obtained by optimizing, in the loops at lines 11 and 20, the constructions of the sets of messages, considering the overlap between the sets. This reduces the time for each loop from $O(nm^2)$ to $O(nm)$, where n is the number of users and items, and m is the number of reviews for each item.

2.5.3 Performance on synthetic data

We evaluated the performance of `vancouver` on simulated data; results on real-world data will be given in Section 4.6. We considered 50 users and 50 items, with each user reviewing 6 items; these numbers are similar to those occurring in our actual assignments. The true quality q_i of each item i we assumed was normal-distributed with standard deviation 1. We assumed that each user j had a characteristic variance v_j, and we let the grade q_{ij} assigned by j to i be

	ρ		σ	
	$k=2$	$k=3$	$k=2$	$k=3$
`avg`	0.82	0.63	0.69	1.21
`vancouver`	0.99	0.93	0.15	0.38

Table 2.2: Performance of `vancouver` algorithm on synthetic data.

equal to $q_i + \Delta_{ij}$, where q_i is the true quality of i, and Δ_{ij} has normal distribution with mean 0 and variance v_j. We assumed that the variances $\{v_j\}_{j \in U}$ of the users were distributed according to a Gamma distribution with scale 0.4, and shape factors $k = 2, 3$. The results are summarized in Table 2.2. For each shape factor, and each of the two algorithms `avg`and `vancouver`, we report the statistical correlation ρ between true quality q_i and consensus quality $\hat{q}_i$ for all items i, as well as the standard deviation σ of the difference $q_i - \hat{q}_i$. Each entry in the table is the average over 100 runs. The `vancouver` algorithm reduces the error between true and consensus grades by a factor between 3 and 4, compared with simple average `avg`. The fact that the gain is larger for shape factor $k = 2$ compared with $k = 3$ indicates that the algorithm performs better when there are fewer, more imprecise users. Even more significant is the increase in the correlation ρ. The code used for the table can be obtained from `https://github.com/lucadealfaro/vancouver`, and corresponds to the tag "2013-techrep"; the code can be easily adapted to study the performance of the algorithms under different sets of assumptions on user behavior.

2.5.4 Performance on the Crowdgrader Data

We performed two different types of evaluations of the precision of the `vancouver` algorithm in assigning consensus grades to assignments. In one type of evaluation, we compared crowdsourced consensus grades with control grades given by the instructor or other domain

experts; in the other type, we measured the grade difference among submissions that we knew were identical.

2.5.4.1 The dataset

The evaluation dataset consisted in five homework assignments for an Android class (CMPS 121); five homework assignments for a C++ class (CMPS 109), and one homework assignment for a Java class (LP2). The number of homework submissions, and reviews, for these classes are summarized in Table 2.3. As the table indicates, students generally performed the reviews that they were asked to do, indicating that the system of incentives we have in place (discussed more in depth in Section 2.6) was effective. Some of the difference between the number of reviews due, and performed, can be ascribed to the fact that students could decline to review specific submissions. The table also shows that, in the initial homework assignments of each class, some submissions received a low number of reviews. This occurred as we had not yet fine-tuned our algorithms for assigning reviews to students. Once we developed algorithms that try to predict the probability that each outstanding review will be completed, we were able to ensure a more uniform review coverage.

2.5.4.2 Evaluation using control grades

For some assignments, we had available control grades given by the instructor, or other domain experts, for a randomly selected subset of submissions that numbered at least 20. For the Android assignments, the control grades were assigned by a Teaching Assistant (TA) who was a fairly accomplished Android developer. For the Java assignment, the control grades

Assignment	$\|S\|$	RevsDue	MinRevs	AvgRevs
CMPS 121 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
CMPS 109 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

Table 2.3: Number of reviews assigned and performed for the homework assignments that are part of the dataset. $|S|$ is the number of submissions, RevsDue is the number of reviews that each student ought to have done, MinRevs is the minimum number of reviews received by a submission, and AvgRevs is the average number of reviews per submission.

were provided by the instructor. For the C++ assignments, the authors graded 20 or more randomly selected submissions for each assignment. We compared the control grades with the consensus grades computed by `avg` and `vancouver` according to the following metrics:

- ρ: the coefficient of statistical correlation (also known as Pearson's correlation) between the control grades $\{q_i\}$ and the consensus grades $\{\hat{q}_i\}$.

- KT: the Kendall-Tau distance between the orderings induced by the control and consensus grades [49]. If r_i and t_i are the ranks received by submission i in the computed, and control, rankings respectively, then $\text{KT} = \sum_i (r_i - t_i)$.

- norm-2: the norm-2 distance $(\sum_i (q_i - \hat{q}_i)^2)^{1/2}$ between the control grades $\{q_i\}$ and the consensus grades $\{\hat{q}_i\}$. Grades were awarded on a scale from 0 to 10 in the assignments.[3]

[3]The grading scale can be chosen for each assignment, but all assignments so far have used a 0 to 10 scale.

- s-score: we first normalize the control grades $\{q_i\}$ and the consensus grades $\{\hat{q}_i\}$, so that they both have zero mean and unit variance, obtaining $\{q'_i\}$, $\{\hat{q}'_i\}$ Then, we compute the standard deviation s of $\{q'_i - \hat{q}'_i\}$, and we report the s-score $1 - s/\sqrt{2}$.

The results for the various assignments are reported in Table 2.4. We see that the results are unclear: in CMPS 121 and JP2, `vancouver` does better; in the two CMPS 109 assignments, it does worse. This may be a consequence of the fact that the primary cause of evaluation error in CMPS 109 consisted in failures encountered by students in compiling the C++ submissions of other students, triggered by development environment (operating system, build chain) differences. These failures are not well modeled by the assumption that each user has an intrinsic review accuracy: the fact that compilation problems occurred in one review may have little bearing on the accuracy of other reviews by the same user. The low correlation between consensus grades and control grades for CMPS 121 is due to the fact that the control grades have a very coarse granularity (few values in the grading scale were used). We also note that this evaluation is inherently approximate, since the control grade is affected by the same type of imprecision that affects the student-provided grades. While instructor and TAs are (usually) more knowledgeable than students in the subject matter, they also make mistakes when grading homeworks, failing to spot problems, or not giving credit to great aspects of the work that go undetected.

2.5.4.3 Evaluation using pairs of identical submissions

For some of the CMPS 109 C++ homework assignments, students were able to work in groups. Since at the time CrowdGrader did not support group submissions (the feature has

Homework	Algorithm	ρ	KT	norm-2	s-score
CMPS 109 hw 2	`avg`	0.75	0.37	1.40	0.50
	`vancouver`	0.69	0.39	1.59	0.45
CMPS 109 hw 3	`avg`	0.84	0.39	1.49	0.60
	`vancouver`	0.80	0.42	1.75	0.55
CMPS 121 hw 3	`avg`	0.39	0.53	1.63	0.22
	`vancouver`	0.49	0.53	1.33	0.29
LP2	`avg`	0.85	0.20	1.75	0.61
	`vancouver`	0.87	0.18	1.79	0.64

Table 2.4: Performance of `avg` and `vancouver`, with respect to control grades.

Assignment	D, `vancouver`	D, `avg`	N. pairs
CMPS 109 hw 2	1.97	3.24	6
CMPS 109 hw 3	1.29	1.39	12
CMPS 109 hw 4	0.98	1.07	20
CMPS 109 hw 5	1.38	1.19	20

Table 2.5: Average square difference between grades received by identical assignments, using crowdsourcing algorithms `vancouver` and `avg`.

since been added), the students were asked to each submit a solution. The student submissions would be graded independently, and the TA, who had a list of groups and their members, would then average the grades received by the students in the same group, and assign to each group member this average. This meant that we had available several pairs of identical submissions, coming from members of the same group. This made it possible to judge the quality of a crowdsourcing algorithm according to how close were the grades received by pairs of such identical submissions. In Table 2.5, we report on the average D of $(\hat{q}_i - \hat{q}_l)^2$, computed over all pairs (i,l) of identical submissions, for the algorithms `vancouver` and `avg`. We see that according to this measure, even for CMPS 109 `vancouver` has generally better performance than `avg`, even though the difference is not large.

2.5.4.4 Discussion

The results presented in this section show that, for our assignments, the `vancouver` algorithm provides a smaller advantage, compared to `avg`, than it would be expected from Table 2.2. We believe that the lower performance is due to the fact that the user error model used in developing algorithm `vancouver`, in which each user i has a variance v_i, is only an approximation for the real behavior of students reviewing submissions. The largest single cause of review errors were:

- Unclear problem statements, that caused different students to have different interpretations of what constituted a good homework solution.
- Variability in the student's code development environment that occasionally prevented students from compiling and evaluating submissions.

The clarity and precision of homework assignments is likely the major factor in the precision of any tool, or any TA, in evaluating submitted solutions. We believe that the higher correlation and quality of the results for the Java assignment are due to the uniformity of the environment enforced for that submission.

We also experimented with a number of variations of algorithm `vancouver`, some based on using notions of median or weighed median for selecting grades. In particular, we experimented with a method we nicknamed "maverage", in which we aggregated student-assigned grades for each item by first discarding the highest and lowest grades, then doing a weighted average using the reciprocal of variance as weights. This process was inspired by the way used to average the grades given by Olympic judges in competitions. We also tried to learn the posi-

tive or negative bias of each student compared to the others, and subtract the bias before using the student's grades. None of these variants was clearly superior to `vancouver`. We believe that larger datasets are needed for us to be able to formulate and validate algorithms superior to `vancouver`.

2.6 Review Incentive and Final Grade Assignment

2.6.1 Review Incentive

To provide an incentive for students to complete a certain number of reviews per assignment, we made the review effort a component of the overall grade that was assigned to students. For each homework assignment, the instructor could choose the number N of reviews each student had to perform, and the fraction $0 < p_r < 1$ of the grade that was due to reviews. Each student j then received for the assignment a *crowd-grade* equal to

$$(1 - p_r)\hat{q}_j + p_r \frac{\min(m_j, N)}{N} \hat{r}_j \ ,$$

where m_j is the number of reviews actually performed by student j, and where $\hat{r}_j$ is the estimated *review quality* of j, which we discuss below. The choice of N and p_r was dictated chiefly by practical considerations. In our coding assignments, evaluating a homework submission entailed a lengthy process of unpacking a submission in its own directory, loading it with a tool, reading the various source code files, compiling it, and testing it sometimes with the help of test data. The whole process would take between 5 and 10 minutes for each homework; we chose

$N = 5$ or $N = 6$, as the results appeared to be sufficiently accurate. We also wanted to ensure that each student was able to learn by reading good-quality submissions by others, and a value of $N = 5$ was sufficient in practice to ensure this (students could always do additional reviews if they wished to see even more solutions). For p_r, a common choice was 0.25, so that 25% of the crowd-grade was due to the reviews. This value roughly reflected the proportion between the time required to review the submissions, and the time required to complete and submit one's own submission.

The decision of how to measure $\hat{r}_j$ for a student j turned out to be more difficult. Initially, we defined it as follows. Let $\{g_{ij}\}_{i \in S, j \in U}$ be the set of all grades that were assigned, and let $\{\hat{q}_i\}_{i \in U}$ be the set of all consensus grades, as before. Then,

$$\tilde{v} = E(\{(g_{ij} - \hat{q}_l)^2\}_{j \in U; i,l \in S})$$

is the average square error with respect to the consensus grades of a hypothetical "fully random" user, who assigns to each submission a grade picked at random from the complete set of assigned grades. The actual average square error $\tilde{v}_j$ of a student $j \in U$ with respect to the consensus grades is instead:

$$\tilde{v}_j = E(\{(g_{ij} - \hat{q}_i)\}_{i \in S}) \ .$$

Therefore, we experimented with assigning to each student a review grade that measured how much better the student was than such a fully random grader, using:

$$\hat{r}_j = 1 - \sqrt{\frac{\min(\tilde{v}_j, \tilde{v})}{\tilde{v}}} \ .$$

This choice appealed to us from a theoretical point of view, especially as it is scale-invariant, so that it would not matter whether students were using the full grading scale (in our case, $[0,10]$) or a subset of it (for instance, assigning grades only in the interval $[4,8]$). However, the choice did not work to our satisfaction in practice. In each assignment, some perfectly honest and motivated students received very low review grades, including 0: strange as it might seem, some students really did worse than a random grader, in spite of their best intentions. Those students were not pleased to see the time they put into reviewing homework submissions go completely unrewarded. The problem was especially acute in the initial homework assignments of each class, where a large fraction of homework submissions received the maximum grade, thereby lowering $\tilde{v}$, and making it harder to improve on the random grader.

As student satisfaction is one of our goals, we needed a different approach. We do not yet have a perfect solution: a metric that is scale invariant and rewards true accuracy as compared to random input, and yet, that students find fair and gratifying. The metric currently used by CrowdGrader is a fairly generous one. We let $v_G = G^2/3.125$ be a reference level for the average square error, where G is the maximum of the grading scale used (we omit the justification, as it is fairly ad-hoc), and we use:

$$\hat{r}_j = 1 - \sqrt{\frac{\min \tilde{v}_j, v_G}{v_G}} \quad .$$

This is not scale-invariant, so that students would get a higher review grade simply by agreeing to use only a small portion of the overall grade range available to them. We are still seeking a scale-invariant solution that students find equitable.

2.6.2 Final grade assignment

CrowdGrader produces *crowd-grades* that depend both on the submission and on the review grades, as described above. The instructor can then either accept these grades as final, or provide final grades for a few of the students; the final grades for the remainder of the students are then derived by interpolation, according to their crowd-grades. This gives the ability to the instructor to re-shape the grade curve of the class. In the Android class (CMPS 121), the instructor relied on this function to manually choose the dividing lines between A/B, B/C, and C/F grades. The instructor examined several assignments chosen from the class rank order, read the reviews, and assign grades (5.3 for A+, 4.5 for the A/B dividing line, etc.) to selected assignments; CrowdGrader then computed the remaining final grades by linear interpolation, in proportion to the crowd-grades. In CMPS 109, the instructors often used the crowd-grades as final grades.

2.7 Error Factors in Peer Grading

Successful peer grading is predicated on the ability to reconstruct a reasonably accurate consensus grade from the grades assigned by the students. This leads to the following question: what factors cause or influence the errors in peer-assigned grades? We are interested in this question for three reasons. First, we wish to obtain a better understanding of the dynamics and human factors in peer grading. Second, a better understanding of the causes of error has the potential to lead to tool improvements that reduce the errors. For example, if mis-understanding on the work submitted constituted a large source of error, then peer grading

tools could be augmented with means for work authors and graders to communicate, so that the misunderstandings could be resolved. Third, a better model of peer grading errors might lead to better algorithms for aggregating the student-assigned grades into the consensus grades for each item.

Our interest in the origin of peer-grading errors is also due to our work on the peer-grading tool CrowdGrader.We have put considerable effort in reducing the error in the consensus grade computed by CrowdGrader, as compared to control instructor-assigned grades. While efforts on the tool UI and UX paid off, as we will detail later, the efforts to create more precise grade-aggregation algorithms did not. In the context of MOOCs, [64] reports a 30% decrease in error using parameter-estimation algorithms that infer, and correct for, the imprecision and biases of individual users. CrowdGrader is used mostly in universities and high-schools. On CrowdGrader data, the parameter-estimation algorithm of [64] offers no benefit compared with the simple "Olympic average" obtained by removing lowest and highest grades, and averaging the rest. Indeed, we have spent a large amount of time experimenting with variations upon the algorithm and new ideas, but we are yet to find an algorithm that offers consistent error reduction of more than 10% compared to the Olympic average. Thus our interest on the origin of errors in CrowdGrader: what are the main causes? What makes them so difficult to remove using algorithms based on parameter estimation, reputation systems, and more?

To gain an understanding of the dynamics of peer grading, we have analyzed a set of CrowdGrader data consisting in 288 assignments, 25,633 submissions, and 113,169 grades and reviews. Of the 25,633 submissions, 2,564 were graded by the instructors in addition to the students. The questions we ask include the following.

Is error mostly due to items or to students? We first ask the question of whether the imprecision in peer grades can be best explained in terms of students being imprecise, or items being difficult to grade. We answer this question in two different ways.

First, we build a parameterized probabilistic model of the review process, similar to the model of [64], in which every review error is the sum of a component due to the submission being reviewed, and of a component due to the reviewer. The parameters of the model are then estimated via Gibbs sampling [35]. The results indicate that students contribute roughly two thirds of the total evaluation error.

This result, however, speaks to the *average* source of error. Of particular concern in peer grading are the very large errors that happen less frequently, but have more impact on the perceived fairness and effectiveness of peer grading. We measure the correlation of large errors in items, and in users; our results indicate that hard-to-grade items are a more common cause of large errors than very imprecise students.

Do better students make better graders? A natural question is whether better students make better graders. In Section 2.7.4 we give an affirmative answer: students whose submissions are in the lower 30%-percentile quality-wise have a grading error that is about 15% above average. The effect is fairly weak, a likely testament to the fundamental homogeneity in abilities in a high-school or college class, as well as to the fact that grading a homework is usually easier than solving the homework.

Does the timing of reviews affect their precision? In Section 2.7.5 we consider the relation of review timing and review precision. We did not detect strong dependencies between grading error and the time taken to complete a review, the order in which the student completed

the reviews, or how late the reviews were completed with respect to the review deadline.

Does error vary with class topic? In Section 2.7.2 we consider the question of whether grading precision varies from topic to topic. Comparing broad topic areas, such as computer science, essays, science, we find the statistics to be quite similar, indicating how general factors are less important than the specifics of each class.

Does tit-for-tat affect review feedback? CrowdGrader allows students to leave feedback on the reviews and grades they receive; this feedback is then used as one of the factor that determines the student's grade in the assignment. The feedback was introduced to provide an incentive for writing helpful reviews. In Section 2.7.6 we show that when a grade is over 20% below the consensus, it receives a low feedback score due to tit-for-tat about 38% of the time.

In the next section we give describe the datasets on which our analysis is based. The subsequent sections present the details of the answers to the above questions. We conclude with a discussion on the nature of errors in peer grading, and on the implications for algorithms and reputation systems for computing consensus grades.

2.7.1 The CrowdGrader dataset

The overall dataset we examined consisted in 288 assignments, for a total of 25,633 submissions and 113,169 reviews, written by 23,762 distinct reviewers. The number of reviewers is smaller than the number of submissions, as some students did not participate in the review phase. Table 2.6 gives a break-down of the dataset according to subject area. On average, each submission received 4.41 reviews, and each reviewer wrote on average 4.76 reviews.

We will refer to submissions also as *items*, and we will refer to students or reviewers

	Assign-ments	Submis-sions	Reviewers	Reviews	Graded Assign-ments	Graded Submis-sions
Computer Science	188	19397	17829	86347	68	2402
Physics	7	274	270	907	6	33
Epidemiology	5	337	313	1551	0	0
Sociology	49	3822	3683	18339	3	16
Business	26	1217	1108	3915	15	106
English	9	397	383	1717	1	7
High-school	7	279	278	1097	5	20
Other	4	189	176	393	0	0
All Combined	288	25633	23762	113169	93	2564

Table 2.6: The CrowdGrader dataset used in this study. *Graded assignments* are the assignments where an instructor or teaching assistant graded at least a subset of the submissions. *Graded submissions* is the number of submissions that were graded by instructors or teaching assistants, in addition to peer grading.

also as *users,* thus adopting common terminology for general peer-review systems.

CrowdGrader includes three features that promote grading accuracy; these features likely influenced the data presented in this study.

Incentives for accuracy. The overall grade a student receives in a CrowdGrader assignment is a weighed average of the student's *submission, accuracy,* and *helpfulness* grades. The *accuracy grade* reflects the precision of the student's grade, compared either to the other grades for the same submission or, when available, to the instructor-assigned grade. The *helpfulness grade* grade reflects the rating received by the reviews written by the student. Combining the submission grade with the accuracy grade creates an incentive for students to be precise in their grading. The amount of incentive can be chosen by the instructor, but the default is to give 75% weight to the submission grade, 15% weight to the accuracy grade, and 10% weight to the helpfulness grade, and most instructors do not change this default.

Ability to decline reviews. Early in the development of CrowdGrader, we noticed that some of the most glaring grading errors occurred when reviewers were forced to enter a grade for submissions that they could not properly evaluate. This occurred, for instance, when students could not open the files uploaded as part of the submission, due to software incompatibilities. To mitigate this problem, we gave students the ability to *decline* to perform reviews of particular submissions. The total number of submissions a student can decline is bounded, to prevent students from "shopping around" for the easiest submissions to review.

Submission discussion forums. Another early source of large errors in CrowdGrader consisted in gross mis-understandings between the author of a submission, and the reviewers. For instance, when zip archives are submitted, the reviewers may expect some information to be contained in one of the component files, whereas the author might have included it in another. Another example consists in mis-organizing the content of a software submission, so that the reviewers do not know how to run it and evaluate it. To remedy this, CrowdGrader introduced anonymous forums associated with each submission, where submission authors and reviewers can discuss any issues they encounter in evaluating the work.

2.7.2 Errors in Peer Grading

Instructor grades and Olympic averages. We measure review error as the difference between individual student grades, and the "consensus grade" for each submission. We consider two kinds of consensus grades. One is the *Olympic average* of the grades provided by the students: this is obtained by discarding the lowest and highest grade for each submission, and taking the average of the remaining grades. The other is the *instructor grade*. In Crowd-

Grader, instructors (or teaching assistants) have the option of re-grading submissions. In some assignments, instructors decided to grade most submissions as control; in other assignments, instructors mostly re-graded only submissions where student grades were in too much disagreement. When considering instructor grades, we consider only assignments of the first type, where instructors graded at least 30% of all submissions. Considering assignments where instructors grade only problematic submissions would considerably skew the statistics. The dataset, for instructor grades, is thus reduced to 19 assignments and 7675 reviews. Instructor and Olympic average grades have a coefficient of correlation $\rho = 0.81$ (with $p < 10^{-200}$), and an average absolute difference of 6.11 on the [0, 100] grading range.

Global and per-topic errors. Table 2.7 reports the size of errors in CrowdGrader peer grading assignments, split by assignment topic, and taking instructor grades and Olympic grades as reference. When the error is measured with respect to instructor grades, computer science, physics, and high-school assignments showed smaller average error than business, sociology and English, all of whose assignments required essay-writing. When the error is measured with respect to Olympic average, is is mainly business and English that show larger error.

2.7.3 Item vs. Student Error

We consider in this section the question of whether error can be attributed predominantly to imprecise students, or to items that are difficult to grade.

	Average Error	N. of Assignments
Computer Science	7.52	15
Physics	10.6	1
Business	16.5	2
English	17.2	1
High School	10.6	1
All	7.67	19

(a) Error with respect to instructor grades, based on assignments with at least 30% of items graded by the instructor.

	Average Error	N. of Assignments
Computer Science	6.34	188
Physics	4.65	7
Epidemiology	4.57	5
Sociology	4.93	49
Business	7.7	26
English	8.37	9
High School	5.09	7
Other	8.15	4
All	6.16	288

(b) Error with respect to Olympic average.

Table 2.7: Mean absolute value difference error by topic. The grading range is normalized to [0, 100].

2.7.3.1 Average error behavior

To compare the contribution of students and items to grading errors, we develop a probabilistic model in which both students and items contribute to the evaluation error. The model is a modification of the $\mathbf{PG}_1$ model in [64], which allowed for student (but not item) error. In our model, each student has a *reliability* and each item has a *simplicity*; the variances of student and item errors are inversely proportional to their respective reliabilities and simplicities.

Precisely:

$$\text{(Reliability) } \tau_u \sim \mathcal{G}(\alpha_0, \beta_0) \text{ for every student } u,$$

$$\text{(Simplicity) } s_i \sim \mathcal{G}(\alpha_1, \beta_1) \text{ for every item } i,$$

$$\text{(True Grade) } q_i \sim \mathcal{N}(\mu_0, 1/\gamma_0) \text{ for every item } i,$$

$$\text{(Observed Grade) } g_{iu} \sim \mathcal{N}(q_i, 1/\tau_u + 1/s_i)$$

$$\text{for every observed peer grade } g_{iu}$$

where $\mathcal{G}(\alpha, \beta)$ denotes the Gamma distribution with parameters α, β, and $\mathcal{N}(q, v)$ denotes the normal distribution with average q and variance v.

Given an assignment, we use Gibbs sampling [35] to infer the parameters $\alpha_0, \beta_0, \alpha_1, \beta_1, \mu_0, \gamma_0$. In order to apply Gibbs sampling, we need to start from suitable prior values for the quantities being estimated. To obtain suitable priors for the distribution of item quality, we first compute an estimated grade for each item using Olympic average, and we obtain μ_0 and γ_0 by fitting a normal distribution to the estimated grades. To estimate prior parameters α_0, β_0 of student reliabilities we fit a Gamma distribution to a set of approximated students reliabilities. In detail, for every student u we populate a list of errors l_u by the student. Again, we computer errors with respect to the average item grades after removing the extremes (the Olympic average). Using the list of error l_u, we estimate a standard deviation σ_u for every student $u \in U$. This allows us to approximate student reliability $\hat{\tau}_u$ as $\frac{1}{\sigma_u^2}$. Prior parameters α_0, β_0 are obtained by fitting a Gamma distribution to the set of estimated student reliabilities $\{\hat{\tau}_u | u \in U\}$. To estimate prior parameters α_1, β_1 for item simplicities we use the same approach as for α_0, β_0; the only

difference is that item simplicities $\hat{s_i}$ are estimated using error lists l_i computed for every item i, rather than for every student u.

	students	items
Average Standard Deviation	14.2	6.4

Table 2.8: The average standard deviation of students and items errors computed over 288 assignment with 25633 items. The grading range is [0, 100].

Table 2.8 reports the average standard deviation of students and items inferred from the model. As we can see, students are responsible for over two thirds of the overall reviewing error.

2.7.3.2 Large error behavior

While students intuitively understand that small random errors will be averaged out, they are very concerned by large errors that, they fear, will skew their overall grade. Thus, we are interested in determining whether such large errors are more often due to students who are grossly imprecise, or items that are very hard to grade. In other words: do large errors cluster more around imprecise students, or around hard-to-grade items? We can answer this question because in CrowdGrader, items are assigned to students in a completely random way. Thus, any correlation between errors on items or students indicates causality.

We answer this question in two ways. First, we measured the information-theoretic *coefficient of constraint.* To compute it, let X and Y be two random variables, obtained by sampling uniformly at random two reviews x and y corresponding to the same item, or to the same student, and letting X (resp. Y) be 1 if x (resp. y) is incorrect by more than a pre-defined

threshold (such as, 20% of the grading range for the assignment). Then, the mutual information $I(Y,X)$ indicates the amount of information shared by X and Y, and the coefficient of constraint $I(X,Y)/H(X)$, where $H(X)$ is the entropy of X, is an information-theoretic measure of the correlation between X and Y.

Tables 2.9 gives $I(X,Y)/H(X)$ for student and item errors, for different values of the error choice, and taking as reference truth for each item either the instructor grade, or the Olympic average for the item. When taking instructor grades as reference (Table 2.9a), large errors are about 5 times more correlated on items than on students, as measured by the coefficient of constraint. When Olympic grades are take as reference (Table 2.9b), large errors are about as correlated on items as they are on students. The difference in behavior is due to the fact that, when an instructor disagrees with the student-given grades on an item, this generates highly correlated errors on that item with respect to the instructor grade, but not with respect to the Olympic average. In any case, the results show that there is no particular correlation on students.

Another way to measure whether large errors tend to cluster around hard-to-evaluate items or around imprecise students consists in measuring the conditional probability $\rho_n = P(\xi \geq n | \xi \geq n-1)$ of an item (resp. student) having $\xi \geq n$ grossly erroneous reviews, given than it has at least $n-1$. If errors on an item (resp. reviewer) are uncorrelated, we would expect that $\rho_1 = \rho_2 = \rho_3 = \cdots$. If these conditional probabilities grow with n, so that $\rho_3 > \rho_2 > \rho_1$, this indicates that the more errors an item (resp. a student) has participated in, the more likely it is that there are additional errors. The values of $\rho_1, \rho_2, \rho_3, \ldots$ allow thus one to form an intuitive appreciation for how clustered around items or students the errors are.

	Error Threshold				
	10%	15%	20%	25%	30%
Students	0.015	0.026	0.017	0.019	0.017
Items	0.075	0.082	0.082	0.1	0. 097

(a) Item errors computed with respect to instructor's grades. We use only assignments that have at least 30% of items grade by the instructor.

	Error Threshold				
	10%	15%	20%	25%	30%
Students	0.018	0.018	0.019	0.020	0.021
Items	0.045	0.030	0.020	0.021	0.020

(b) Item errors computed with respect to Olympic average.

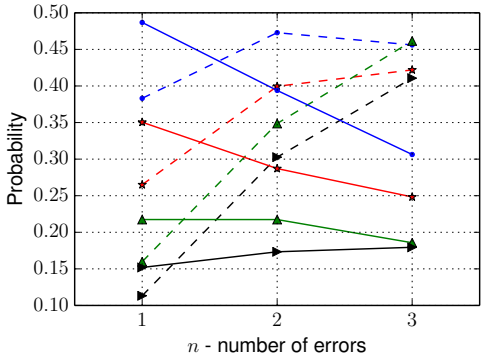
Table 2.9: Coefficient of constraint $I(X,Y)/H(X)$ of large errors on the same item or by the same student, for different error thresholds.

The results are given in Figure 2.1. The data shows some clustering around users, for large errors of over 30% of the grading range. However, clustering around users seems weaker than clustering around items.

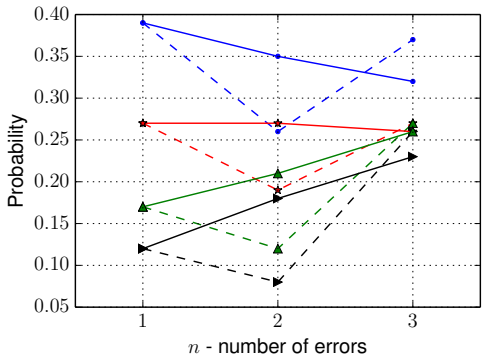
This provides a possible explanation for why reputation systems have not proved effective in dealing with errors in peer-graded assignments with CrowdGrader. Reputation systems are effective in characterizing the precision of each student, and taking it into account when computing each item's grade. Our results indicate however that errors in CrowdGrader are not strongly correlated with students, limiting the potential of reputation systems.

2.7.4 Student ability vs. accuracy

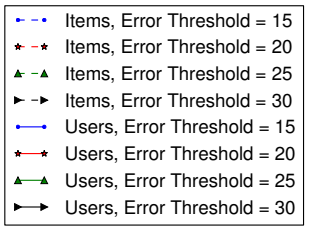
A natural question is whether better students make better graders. To answer this question, we can approximate the expertise of every student with the grade received by the student's own submission, and we can then study the correlation between the student's submis-



(a) Errors computed with respect to the instructor's grades. We use only assignments that have at least 30% of items grade by the instructor.



(b) Errors computed with respect to Olympic average.



(c) Legend

Figure 2.1: Conditional probabilities $\rho_n = P(\xi \geq n | \xi \geq n-1)$ of least n errors given at least $n-1$ errors. We considered error thresholds of 15%, 20%, 25%, 30%.

sion grade, and the review error. As we have only partial coverage of students with instructor grades, we compute the grade received by the student's own submission via Olympic average, rather than instructor grade. As the two generally are close, this increases coverage with minimal influence on the results. We study grading error with respect to both instructor grades and Olympic average.

2.7.4.1 Aggregating data from multiple assignments

When aggregating data from multiple assignments, we cannot directly compare absolute values of grades, or absolute amount of time spent reviewing: each assignment has its own grade distribution, review time distribution, and so forth. To account for variation across assignments, we use the following approach. For each student there is an independent variable x, and an error e. In this section, x is the grade received by the student's own submission, measured via Olympic average; in the next section, x will be related to the time spent during the review, or the time at which the review is turned in. The error e is the difference, for each review, between the grade assigned as part of the review, and the grade of the reviewed submission, obtained either via Olympic average or via instructor grading.

First, for each assignment independently, we sort all students according to their x-value, and we assign them to one of 10 percentile bins: if the assignment comprises m students and the student ranks k-th, the student will be in the $\lceil 10k/m \rceil$ bin; we call these bins the 10%, 20%, ..., 100% bins. For each assignment a, we normalize the grading range to $[0,100]$, and we let $n_{a,q}$ and $e_{a,q}$ be the number of students and the average error in the q percentile bin of assignment a, respectively. The average error for assignment a overall is thus $e_a = \sum_q n_{a,q} e_{a,q} / \sum_q n_{a,q}$. There are two ways of measuring the average error $e_{a,q}$ for one bin: as average absolute value error, or as average root-mean-square error. The two approaches lead to qualitatively similar conclusions, as we show later in this section. We present here only the results for average absolute value, as they are somewhat less sensitive to rare large errors, and thus, more stable.

We aggregate data from multiple assignments, computing for each percentile bin an absolute and a relative error, as follows. The *absolute* error e_q for each percentile q is computed as

$$e_q = \sum_a n_{a,q} e_{a,q} \Big/ \sum_a n_{a,q}. \tag{2.1}$$

The *relative* error r_q for each percentile q is computed as

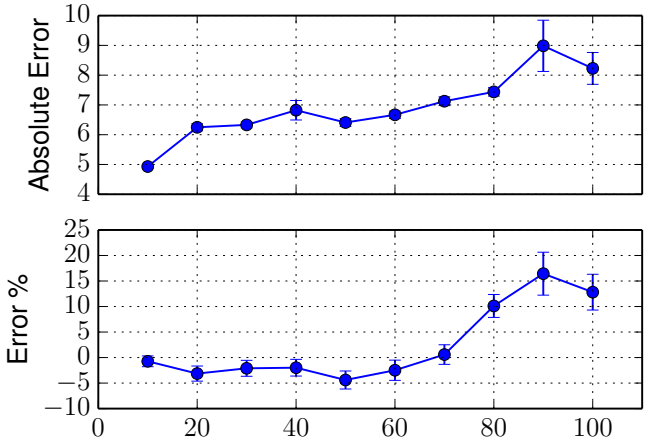
$$r_q = \sum_a n_{a,q} \left(e_{a,q} / e_a \right) \Big/ \sum_a n_{a,q}, \tag{2.2}$$

where $e_{a,q}/e_a$ is the relative error of bin q in assignment a.

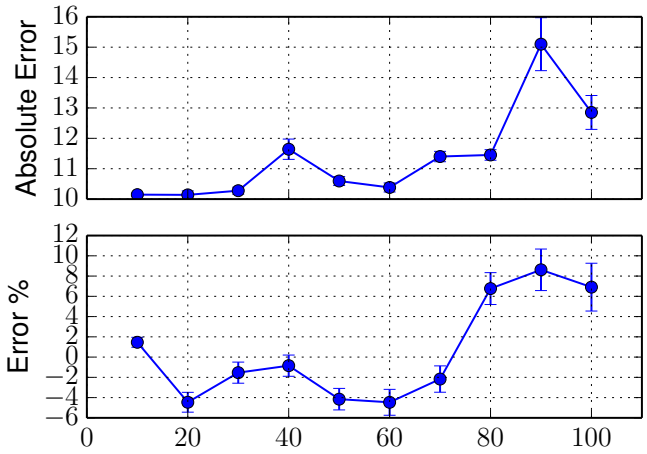
2.7.4.2 Student ability vs. error

The data reported in Figure 2.2b shows the existence of some correlation between student submission grade, and grading precision, measured with respect to the Olympic average. In relative terms, students in the 80–100% percentile brackets show error that is 10% to 20% greater than students with higher submission grade. The absolute error tells a similar story. The two graphs do not have the same shape, due to the fact that relative errors are computed in (2.2) in a per-assignment fashion. In Figure 2.2a we report the same data, computed using rms error rather than average absolute value error. The data is qualitatively similar. In the remaining graphs we consider only average absolute error.

In Figure 2.3 we compare the error with respect to Olympic average with the error compared to instructor grades, for the subset of classes where at least 30% of submissions have been instructor-graded. While the absolute values are different, we see that the curves are very



(a) Mean absolute value difference error.



(b) Root mean square error.

Figure 2.2: Average grading errors arranged into authors' submissions quality percentiles. Grading errors and submission qualities are measured with respect to the Olympic average grades. The first percentile bin 10% corresponds to reviewers that have authored submissions with highest grades. Error bars correspond to one standard deviation.

closely related, indicating that Olympic averages are a good proxy for instructor grades when studying relative changes in precision. The error with respect to instructor grades has very wide error bars for the 90% percentile, mainly due to the low number of data points we have for that

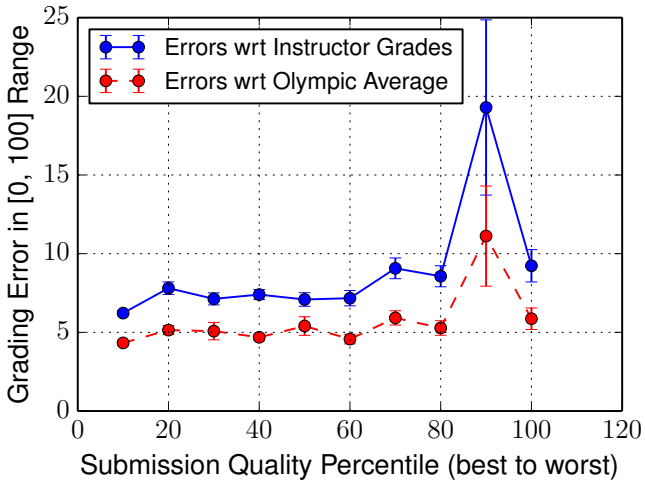


Figure 2.3: Average grading error arranged into authors' submission quality percentiles. The first percentile bin 10% corresponds to reviewers that have authored submissions with highest grades. We report the error both with respect to instructor grades, and to the Olympic average, considering only assignments for which at least 30% of submissions have been graded by instructors. Error bars correspond to one standard deviation.

percentile bracket in our dataset. We favor the comparison with the Olympic average, since the abundance of data makes the statistics more reliable.

The correlation between student ability (as measured by the submission score) and grading precision is lower than we expected. This might be a testament to the clarity of the rubrics and grading instructions provided by the instructors: apparently, such instructions ensure that most students are able to grade with reasonable precision the work by others. This may also be a consequence of the fundamental skill and background homogeneity of students in a classroom, as compared to a MOOC. We note that [8] also reported low correlation between student grades and student precision in the related setting of peer instruction.

2.7.5 Review timing vs. accuracy

We next studied the effect of the time taken to perform the reviews, and the order in which they were performed, on review accuracy. These measurements are made possible by the fact that CrowdGrader assigns reviews one at a time: a student is assigned the next submission to review only once the previous review is completed. This dynamic assignment ensures that all submissions receive a sufficient number of reviews. If each student were pre-assigned a certain set of submissions to review, as is customary in conference paper reviewing, then students who omitted or forgot to perform reviews could cause some submissions to receive insufficient reviews. CrowdGrader records the time at which each submission is assigned for review to a student, and the time when the review is completed. For these results, to conserve space, we provide the error only with respect to the Olympic average, for which we have more data. A comparison of error with respect to Olympic average and instructor grades confirms that the Olympic average is a good proxy for studying variation with respect to instructor grade.

Time to complete a review. We first considered the correlation between the time spent by students performing each review, and the accuracy of the review; the results are reported in Figure 2.4. The results indicate that reviews that are performed moderately quickly tend to be slightly more precise. The correlation is weaker than we expected. We expected to find error peaks due to students that spent very little time reviewing, and that entered a quick guess for the submission grade, rather than performing a proper review. There are no such peaks: either students are very good at quickly estimating submission quality, or they mostly take reviewing and seriously in CrowdGrader. We believe the latter hypothesis is likely the correct one: for

instance, in many computer science assignments, there is no good way of "eye-balling" the quality of a submission without compiling and running it.

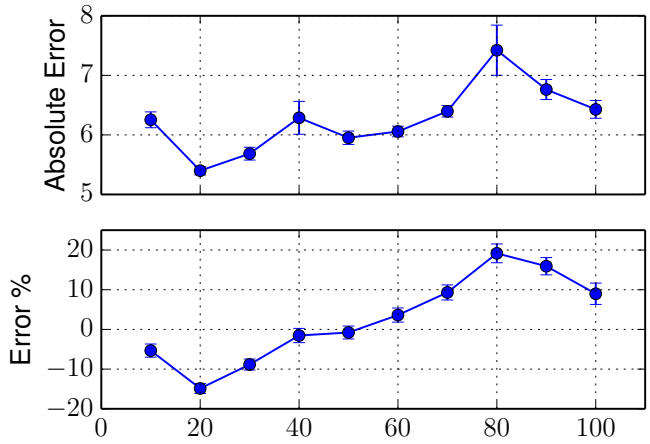


Figure 2.4: Absolute and relative grading error vs. the time employed to perform a review; the first percentile bin 10% corresponds to reviews with shortest review time. The grading range is normalized to $[0, 100]$, and the error is measured with respect to the Olympic average. The error bars indicate one standard deviation.

Time at which a review is completed. Next, we studied the correlation between the absolute time when reviews are performed, and the precision of the reviews. Figure 2.5 shows the existence of a modest correlation: the reviews that are completed in the first 10% percentile tend to be 10% more accurate than later reviews. The effect is rather small, however. In a typical CrowdGrader assignment, students are given ample time to complete their reviews, and the reviews themselves take only one hour or so to complete. Students likely do not feel they are under strong time pressure to complete the reviews, and time to deadline has little effect on accuracy.

Order in which reviews are completed. Lastly, we study whether the order in which a student performs the reviews affects the accuracy of the reviews. We are interested in the ques-

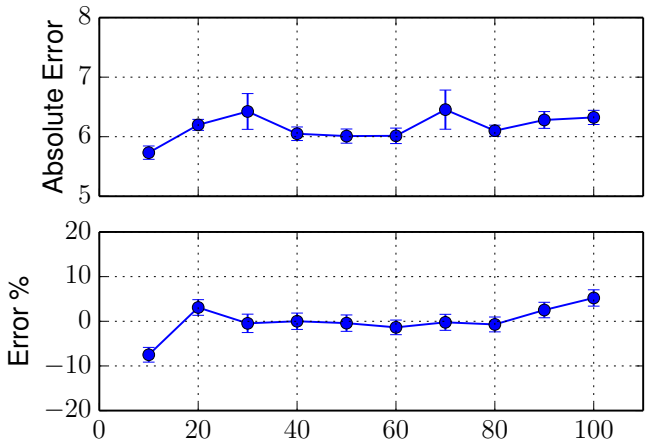


Figure 2.5: Absolute and relative grading error vs. absolute time when a review is completed. The first percentile bin 10% corresponds to the 10% of reviews that were completed first among all assignment reviews. The grading range is normalized to $[0, 100]$, and the error is measured with respect to the Olympic average. The error bars indicate one standard deviation.

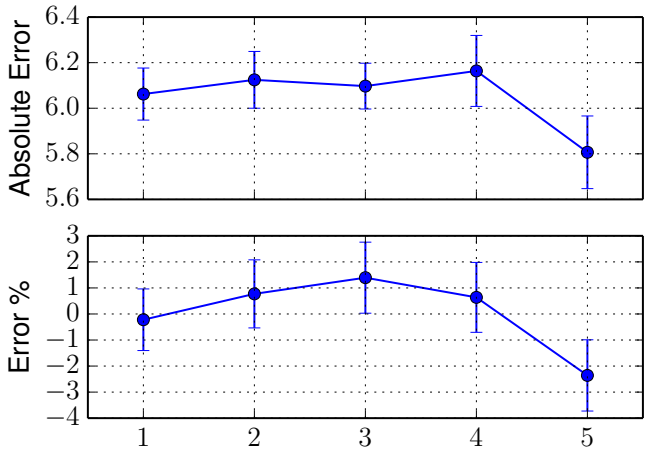


Figure 2.6: Absolute and relative grading error vs. ordinal number of a review by a student. The review 1 is the first a student performs, 2 is the second, and so forth. The grading range is normalized to $[0, 100]$, and the error is measured with respect to the Olympic average. Error bars indicate one standard deviation.

tion of whether students learn while doing reviews, and become more precise, or whether they grow tired and impatient as they perform the reviews, and their accuracy decreases. Figure 2.6 shows that the accuracy of students does not vary significantly as the students progress in their review work. Evidently, the typical review load is sufficiently light that students do not suffer from decreased attention while completing the reviews.

2.7.6 Tit-for-tat in review feedback

In CrowdGrader, students can leave feedback to each review and grade they receive. The feedback is expressed via 1-to-5 star rating systems as follows:

- 1 star: factually wrong; bogus.
- 2 stars: unhelpful.
- 3 stars: neutral.
- 4 stars: somewhat helpful.
- 5 stars: very helpful.

Many such ratings are given as tit-for-tat: when a student receives a low grade, the student responds by assigning a low feedback score (typically, 1 star) to the corresponding review. Indeed, CrowdGrader includes a technique for identifying such tit-for-tat, so that students, whose overall grade depends also on the helpfulness of their reviews, are not unduly penalized. We were interested in analyzing the question of how prevalent tit-for-tat is.

Overall, review grade and review feedback have a correlation of 0.39, with a p-value smaller than 10^{-300}. The correlation between grade and feedback indicates tit-for-tat, as there

is no reason why lower grades should per-se be associated with written reviews that are less helpful. Interestingly, the correlation is fairly independent from the subject area. To bring the tit-for-tat into sharper evidence, we computed also the following statistics. We consider a grade a p (resp. n) outlier if the grade is over 20% above (resp. below) the Olympic average. We then measured the conditional probabilities P_p, P_n that p and n outliers would receive a one or two-star rating, conditioned over the probability that the reviews received a rating at all (students do not always rate the reviews they receive). Over all assignments, we measured $P_p = 0.06$ and $P_n = 0.44$. Since there is no a-priori reason why overly negative reviews may be of worse quality than overly positive ones, the excess probability $P_n - P_p = 0.38$ can be explained by tit-for-tat. This shows that tit-for-tat is rather common: for grades that are 20% or more below the consensus, there is a 38% probability of low feedback due to tit for tat. Fortunately, it is easy to discard low ratings given in response to below-average grades, as CrowdGrader does.

2.8 Conclusions

We conclude with some informal impressions on the performance of CrowdGrader in a class setting.

We investigated many cases where the control and consensus grades differed by some non-trivial amount. In some cases, this was due to superficial reviews by students using CrowdGrader. However, in other cases the problem was with the control grade, as the instructor or TA had missed problems with the submission that were instead detected by some students reviewing it. Overall, for coding assignments, our impression was that the consensus grades computed

by CrowdGrader were at least of the same quality as those provided by a TA. A TA is more consistent in evaluating submissions, paying attention to the same aspects of each submission. On the other hand, the greater number of reviews used in CrowdGrader led to a more comprehensive assessment, in which flaws or positive aspects were more likely to be pointed out. From the perspective of the individual student, we felt the two grading options were of similar quality: with TAs, the risk is that they do not pay attention in their grading to the aspects where most effort is put (or where the flaws are); with crowdsourced grades, the risk is in the inherent variability of the process.

Where the crowdsourced evaluations proved clearly superior was in the feedback provided to the students. When instructors or TAs are faced with grading a large number of assignments, the feedback they provide on each individual assignment is usually limited. With CrowdGrader, students had access to multiple reviews of their homework submissions.

In coding assignments, there is usually more than one way to solve each problem, and students commented on the benefit of being able to see, and learn from, other students' solutions. Students who could not complete the assignment particularly benefited from being able to examine several different working solutions to the homework problems.

In informal comments we received, the two aspects of CrowdGrader students appreciated the most was the quality of the feedback received, and the ability to learn from other students' solutions. The one aspect they enjoyed the least, of course, was the time it took for them to do the reviews.

While CrowdGrader may not be suitable for all types of homework assignments, the tool performed to our satisfaction for coding assignments, and we believe that the tool is well-

suited to any homework assignment where students can, by comparing solutions among them and with their own, come to an assessment of their peers' work.

We also presented an analysis of a large body of peer-grading data, gathered on assignments that used CrowdGrader across a wide set of subjects, from engineering to business and humanities. Our main interest consisted in identifying the factors that influence grading errors, so that we could devise methods to control or compensate for such factors. Out results can be thus summarized:

- Large errors are no more strongly correlated on students than they are on items. In other words, students who are imprecise on many submissions are not a dominant source of error.

- There is some correlation between the quality of a student's own submission (which is an indication of the student's accomplishment), and the grading accuracy of the student, but the correlation is weak and limited to the student with highest, and lowest submission grades.

- There is little correlation between the accuracy of a review, and the time it took to perform the review, or how late in the review period the review was performed.

- There is clear evidence of tit-for-tat behavior when students give feedback on the reviews they receive.

All of the correlations we measured, except for the tit-for-tat one, are rather weak. This is a reassuring confirmation that peer-grading works as intended.There are no large sources of

uncontrolled error due to factors such as student fatigue in doing the reviews, or gross inability of weaker students to perform the reviews. The peer-grading tool, in our classroom settings, ensures that the remaining errors are fairly randomly distributed, with little remaining structure.

The results highlight the difficulties in using reputation systems to compute submission grades in peer-grading assignments in high-school and university settings. Reputation systems characterize the behavior of each student, in terms for instance of their grading accuracy and bias, and compensate for each student's behavior when aggregating the individual review grades into a consensus grade. However, our results indicate that the large errors that most affect the fairness perception of peer grading are most closely associated with items, rather than with students. Reputation systems are powerless with respect to errors caused by hard-to-grade items: even if they can correctly pinpoint which submissions are hard to grade, little can be done except flagging them for instructor grading. Indeed, the reputation system approach of [64], which yielded error reductions of about 30% for MOOCs, yielded virtually no benefit in our classroom settings.

There is more potential, instead, in approaches that make it easier to grade difficult submissions. In CrowdGrader, we introduced anonymous forums, associated with each submission, where submissions authors and reviewers can discuss any issues that arise while reviewing the submission. These forums are routinely used, for instance, to solve the glitches that often arise when trying to compile or run code written by someone else. Anectodally, these forums have markedly increased the satisfaction with the peer-grading tool, as students feel that they have a safety net if they make small mistakes in formatting or submitting their work, and are in the loop should any issues occur.

Chapter 3

Incentives Schemes for Truthful Evaluations

3.1 Problem Setting and Contributions

Crowdsourcing allows access to large populations of human workers, and it can be an efficient and cheap solution for many applications. The very farming out of work to many independent workers, however, creates the problem of quality control. In the absence of effective supervision or quality-control mechanisms, the workers may submit low quality work, or they may deliberately engage in straight-out vandalism. Workers can also collude with each other to game the system and collect rewards without performing the required work. In this paper, we describe supervisory schemes that provide an incentive towards high-quality work, and we show that the incentive is both cheap in terms of the required supervisor time and work overhead, and effective in making honest and accurate work the best strategy for workers.

We focus on crowdsourcing tasks which are *verifiable*, that is, they have objective answers that a supervisor or another worker can check to conclude whether a worker is submitting

quality work or not. We further consider two types of verifiability: *binary,* and *quantitative.* In binary verifiable tasks, the question of whether a worker submits quality work can be answered with either a Yes or a No. In quantitatively verifiable tasks, the solution of a task is a real number, and we can measure the quantitative difference between the submitted and the true solutions of tasks. Classification tasks are examples of binary verifiable tasks, as supervisors can check that the classification in discrete categories submitted by a worker matches expectations. Grading tasks are examples of quantitatively-verifiable tasks.

We propose schemes that provide truthful incentives to the human workers at a low cost for the supervisors regardless of the size of the human worker population.

Using golden sets is a practice for quality control in crowdsourcing [78]. Golden sets are sets of tasks for which the answer is known to the supervisor and are presented to workers with the goal of evaluating their performance; such sets showed a positive impact on worker performance in crowdsourcing systems [42]. Golden sets, however, can be difficult and costly to obtain [62], as they create an overhead for workers to perform extra tasks for quality control. In applications that already require significant incentive to extract a small amount of items of work from workers, wasting work on golden set tasks is undesirable if there is an alternative. Moreover, golden sets might be unavailable in advance. For example, in peer grading of homework assignments, a golden set would need to be constructed for each homework assignment (unless the assignments are identical). In addition, the golden set approach is problematic from a mechanism design point of view: knowing that the supervision is performed in this way, workers can infer the identity of golden set tasks by intersecting allocated tasks of different workers, and minimize effort by only being truthful in the golden tasks. For class homework, where

students can communicate with each other and some distinctive features may be easy to spot ('Did you also have to grade the Android app where the ball goes through the green wall?'), the golden set approach can be particularly weak.

The schemes that we propose rely on comparing the answers given by workers performing the same tasks with each other; in particular, they do not require golden sets of tasks for which the answer is known in advance.

We first study a simple *one-level* scheme and thereafter we propose a *hierarchical* scheme.

In the simple *one-level* scheme, workers perform tasks which are directly evaluated by the supervisor with some probability. We study the conditions that ensure that workers maintain the incentive to provide truthful answers.

The one-level scheme does not scale to large crowds, as the supervisor needs to perform an amount of work that grows linearly with the number of workers. Thus, we introduce a *hierarchical* scheme, where the work of the supervisor is bounded even as the number of tasks and workers grows. The scheme organizes workers in a hierarchy, where the supervisor is at the top, and the other workers are arranged in layers below. Every worker in the hierarchy shares one common task with each worker below, so that it can verify part of the work performed by lower levels of the hierarchy. This hierarchical verification scheme entails no wasted work, and provides a truthful incentive to the workers regardless of their level in the hierarchy. The scheme is based on one, uniform, category of workers: we do not need to split workers into "regular" workers and meta-reviewers. As the worker population increases, the hierarchy becomes deeper, but the amount of work that the supervisor needs to do remains constant, and so

does the incentive towards correct behavior. We show that the only information about the hierarchy that needs to be communicated beforehand to the workers is their level in the hierarchy itself. We provide matching upper and lower bounds for the amount of information that needs to be communicated beforehand to workers in the hierarchy to maintain a truthful incentive, showing that a logarithmic amount of information in the number of workers is both necessary and sufficient.

We study the practical aspects of the implementation of the hierarchy. Many crowdsourcing tasks benefit from redundancy, that is, from assigning the same task to more than one worker. For instance, by assigning the same item to multiple graders, it is possible to reconstruct a higher-accuracy grade for the item than would be available from one grader alone [65] We show that in redundant tasks in which there is no control over task allocation to workers, the problem of creating an optimal hierarchy is NP-hard. We present fast approximation algorithms that are optimal within constant factors. If the supervisor can control the allocation of tasks to workers, as in many real applications, we show that constructing the hierarchy is an easy problem.

We develop our results first in the case of binary verifiable tasks. These are common in classification tasks: spam or not, correct answer or not, etc. We consider a model where workers need to make an "effort" of $f(e)$ in order to ensure that their error probability is lower than e. We obtain a tight lower bound for the mistake penalties necessary to ensure that the correctness incentive propagates to all levels of the hierarchy. We show that the truthful incentive holds even when the supervisor occasionally makes mistakes, and in populations of workers with diverse proficiency, where workers can have limited proficiency, provided that there are enough

proficient workers in the crowd.

We then show how the results on binary verifiable tasks transfer to the case of quantitative tasks. In quantitative tasks, the notion of a task performed correctly is replaced by the notion of variance in the quantitative outcome of the task. The effort function relates the effort (or cost) to the worker to the variance in the worker's evaluation. In the model, increased worker effort produces higher expected precision of the worker's answers, and is similar to other models proposed in the literature [12]. We show that we can shape the incentives to ensure that it is optimal for all players to put sufficient effort to ensure their variance is below a given threshold, independently of the worker position in the hierarchy. In other words, hierarchical distance from the supervisor does not entail loss of precision in the tasks performed. This enables the scheme to scale to arbitrarily large crowds, while keeping the work of the supervisor bounded and the precision constant.

The proposed schemes are thus applicable to a multitude of crowdsourcing applications, from conventional classification tasks using generic crowds in crowdsourcing marketplaces to peer grading in Massive Open Online Courses with an arbitrarily large population of students.

3.2 Example of Worker Collusion in a Peergrading Setting

Reviewing is hard work. In order to motivate students to perform high quality reviews of other students' work, some incentive is needed. A simple approach consists in making the review work part of the overall assignment grade, giving each student a review grade that is

related to the student's grading accuracy. To measure the grading accuracy of a student, the simplest solution is to look at the discrepancy between the grades assigned by the student, and the consensus grades computed from all input on the assignment.

Unfortunately, such approach opens up an opportunity for students to game the system. A big enough group of students can affect the consensus grades and thus affect how they and other reviewers are evaluated. One obvious grading strategy for a reviewer is to assign the maximum grade to every assignment they grade. In this way, students spend no time examining the submissions, and yet get perfect grades both for their submission, and for their reviewing work.

We have observed this behavior in real classrooms. In a class whose grading data we analyzed, held at a US university,[1] the tool CrowdGrader[2] was used to peer-grade homework. The initial homework assignments were somewhat easy, so that a large share of submissions deserved the maximum grade on their own merit. As more homework was assigned and graded, a substantial number of students switched to a strategy where they assigned the maximum grade to every submission they were assigned to grade. Submissions that had obvious flaws were getting high grades, and reviewers who did diligent work were getting low review grades because their accurate evaluations did not match the top-grade consensus for the submissions they reviewed. Figure 3.1 displays the fraction of students who assignmed maximum grades to assignments in the class. A surprisingly high percentage of students were giving maximum grades; the percentage rose to 60% in the 13th assignment. Between the 13th and 14th assignment there was a big drop in the fraction of such students, as the instructor announced that there

[1] Privacy restrictions prevent us from disclosing more details on the class.
[2] www.crowdgrader.org

would be a new grading procedure introduced that would penalize such behavior. However, the hastily-introduced procedure did not work, and the students returned to give inflated evaluations spending little time reviewing.

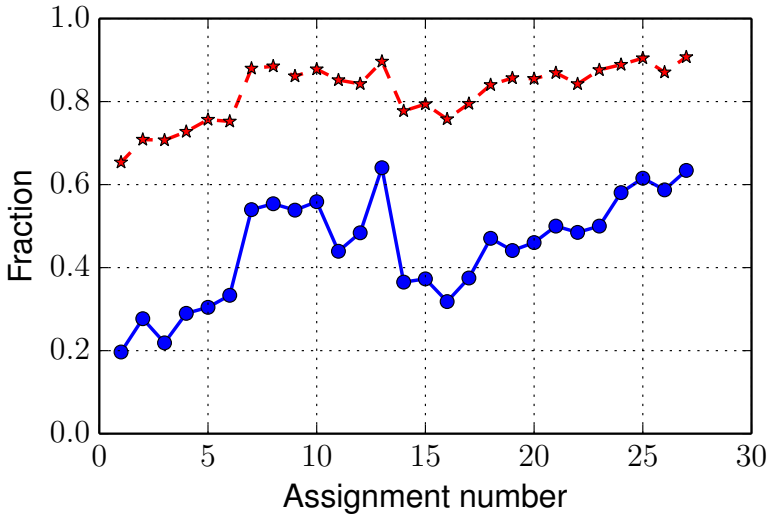


Figure 3.1: Frequency of assignments receiving maximum grades for a class with 27 homework assignments and 83 students; each student graded 5 homework submissions. The dashed line plots the number of maximum grades, as a fraction of all grades assigned, for each homework. The solid line plots the fraction of students who gave maximum grades to all the submissions they graded, for each homework.

3.3 Related Work

Providing incentives to human agents to return truthful responses is one of the central challenges for crowdsourcing algorithms and applications [36].

Prediction markets are models with a goal of obtaining predictions about events of interest from experts. After experts provide predictions, a system assigns a reward based on a

scoring rule to every expert. Proper scoring rules ensure that the highest reward is achieved by reporting the true probability distribution [85, 44, 17]. An assumption of the scoring rules is that the future outcome must be observable. This assumption prevents crowdsourcing systems to scale to large crowds as obtaining the correct answer for each event or task is prohibitively expensive.

The model presented in [15] relaxes this assumption. The proposed scoring rule evaluates experts by comparing them to each other. The model assigns a higher score for an expert if her predictions are in agreement with predictions of other experts. Work [15] belongs to the class of peer prediction methods. Peer prediction methods is wide class of models for providing incentives [66, 58, 28, 87, 86, 20, 46, 68, 72, 90, 79, 47, 51] . Such methods elicit truthful answers by analyzing the consensus between workers in one form or another. Peer prediction methods encourage cooperation between workers and, as a result, promote uninformative equilibria. The study in [45] shows that for the scoring rules proposed in the peer-prediction method [58], a strategy that always outputs "good" or "bad" answer is a Nash equilibrium with a higher payoff than the truthful strategy. Works by [46, 79] show that the existence of such equilibria is inevitable. In contrast, hierarchical incentive schemes we propose make the truthful strategy the only Nash equilibrium.

The model described in [46] considers a scenario of rational buyers who report on the quality of products of different types. In the developed payment mechanism the strategy of honest reporting is the only Nash equilibrium. However, the model requires that the prior distribution over product types and condition distributions of qualities is the common knowledge. This requirement is a strong assumption.

The Bayesian Truth Serum scoring method proposed in [66] elicits truthful subjective answers on multiple choice questions. The author shows that the truthful reporting is a Nash equilibrium with the highest payoff. The model is different from other approaches in that besides the answers, workers need to provide predictions on the final distribution of answers. Workers receive a high score if their answer is "surprisingly" common - the actual percentage of their answer is larger than the predicted fraction. Similarly, incentive mechanisms in [87, 86, 68, 69, 72] require workers provide belief reports along with answers on tasks. Truthful mechanisms in [58, 90, 51] requires knowledge about the distribution from which answers are drawn. Our mechanisms do not rely on worker's beliefs on other workers' responses nor require knowledge about the global answer distribution.

The work in [5] studies the problem of incentives for truthfulness in a setting where persons vote other persons for a position. The analysis derives a randomized approximation technique to obtain the higher voted persons. The technique is strategyproof, that is, voters (which are also candidates) cannot game the system for their own benefit. The setting of this analysis is significantly different from ours, as the limiting assumption is that the sets of voters and votees are identical. Also, the study focuses on obtaining the top-k voted items, while in our setting we do not necessarily rank items.

The PeerRank method proposed in [80] obtains the final grades of students using a fixed point equation similar to the PageRank method. However, while it encourages precision, it does not provide a strategyproof method for the scenario that students collude to game the system without making the effort to grade truthfully.

Authors of [62] propose an automated process to generate golden tasks for quality

assurance in crowdsourcing. An initial set of golden tasks is used to bootstrap a larger set of golden tasks. A tasks is chosen if it has several matching answers by the reliable workers, that is, workers who provided correct answers to the original golden tasks. The chosen tasks are then used to create new golden tasks by injecting common errors. The step of error injecting is to ensure that common error types are present in then new golden set. Note that the process of detecting common error types is manual. The authors report on decreasing amount of manual work to manage large crowds. The main difference with our work is that we provide theoretical guarantees that the proposed incentive mechanisms require constant amount of work by the supervisor for arbitrary large crowds. Moreover, golden sets are not suitable for all applications. For example, in peergrading of homework assignments the total set of homework submissions cannot be obtained before the homework is posted. Also, information on the competence of workers from previous homeworks or classrooms cannot be used reliably for newer homeworks or in different classrooms with different material. The incentive schemes we propose do not require golden sets.

Employees in organizations and firms are frequently organized into hierarchies. Economists study incentives in hierarchical organizations, the influence of hierarchical structure on firms sizes and the loss of control within hierarchies. Previous studies on hierarchies relevant to ours are found in [84, 13, 14, 67]

However, our work and the work by organizational economists is not directly comparable due to different models. Our model is designed to reflect the nature of evaluation tasks. Models by [84, 13, 14, 67] are designed to reflect economic aspects of firms. Our work is not directly comparable to the work of organizational economist, as the model we describe are ap-

plicable to peer grading setting rather than corporate hierarchy setting. For example, the model described in [84] assumes that workers on the bottom layer do the production work, while all other workers (managers) do the coordination and supervision work. Subordinate workers satisfy requests by their superiors with a discount factor that is within $(0, 1)$ range. The smaller the discount factor, the smaller the contributions by the workers to the firm's revenue. In contrast, in our models, all workers perform evaluation work. Workers are evaluated based on the comparison of their answers to the answers of their supervisors. Our models admit that worker can make mistakes or have bounded proficiencies. The work in [84] shows that there is a limit on the size of hierarchy due to loss of control. In contrast, the hierarchies of workers that we propose have the property that the incentive to do truthful evaluations does not deteriorate with the hierarchy depth.

3.4 Crowdsourcing Models

We consider two crowdsourcing models: the binary-verifiable model and the quantitatively-verifiable model. In the binary-verifiable model, tasks can be done either correctly, or incorrectly. For example, a task of classifying items from a discrete set categories is binary-verifiable. In the quantitatively-verifiable model, the solution to tasks are real numbers, and we can measure the distance of the answers given from the correct answer. The tasks of assigning a real-valued grade to a homework submission is an example of a quantitatively-verifiable tasks. We make these settings precise via the following models.

Let U and I be the set of workers and tasks respectively. Every worker u performs a

subset of tasks from I. We construct a bipartite graph $G = (U \cup I, E\}$ with tasks and workers as nodes. For a task $i \in I$ and a worker $u \in U$, the edge (i,u) belongs to the set of edges E iff the worker u was assigned the task $i \in I$. We denote the set of tasks assigned to a worker u as ∂u, the set of workers assigned with a task i as ∂i.

3.4.1 The Binary-verifiable Model

In the binary-verifiable model, each task $i \in I$ has a solution from a set A. A worker performs task i by choosing a solution from set A. Every tasks $i \in I$ has a correct solution $s_i \in A$. To find the correct solution the worker needs to make effort.

Effort Function. Let a_i be a solution proposed by a worker on a task $i \in I$, and let e be the probability that a_i is wrong, i.e. $e = Pr(a_i \neq s_i)$. The effort is defined by a function $f : (0,1] \to [0,+\infty)$, so that a worker needs to pay cost $f(e)$ to have error probability at most e. We require the cost function f to be monotonically decreasing (larger error bounds cost less), and to be differentiable and strictly convex. Requiring that f is convex does not entail any loss in generality. For $x, y \in (0,1]$ and $0 < \alpha < 1$, if we had $f((1-\alpha)x + \alpha y) > (1-\alpha)f(x) + \alpha f(y)$, contradicting convexity, then the worker would obtain a lower cost simply by paying $f(x)$ a fraction $1-\alpha$ of the time, and $f(y)$ a fraction α of the time, obtaining overall error probability equal to $(1-\alpha)x + \alpha y$ at a cost lower than $f((1-\alpha)x + \alpha y)$. Strict convexity of f entails that, the closer x goes to 0, the more difficult it gets to reduce the error by the same amount.

Strategies. A worker's strategy is the choice of error probability e and corresponding effort $f(e)$. For a specified error threshold $\varepsilon \in (0,1]$, we call a strategy with error probability e truthful iff $e < \varepsilon$.

Supervision. We assume that there is a supervisor who can verify whether tasks are done correctly or not. Let worker u provides a solution a_u on a task i, while the supervisor provides a solution a for the same task. The supervisor assigns loss $l(a_u, a)$ to the worker defined by $l(a_u, a) = 0$ if $a_u = a$, and $l(a_u, a) = C$ if $a_u \neq a$, for a fixed punishment cost $C > 0$. Note that, when the supervisor verifies a task i, the supervisor can verify all the workers that also performed i, that is, all the workers in ∂i.

3.4.2 The Quantitative Model

In the quantitative model, the workers are asked to evaluate items from a set I, associating to each item $i \in I$ a value $q_i \in \mathbb{R}$.

Effort Function. In order to produce a precise measurement of the quality of an item, a worker needs to pay a price defined by an effort function. To produce a measurement of the quality of an item to within variance v, a worker needs to pay a price $f(v)$, where f is a non-negative, monotonically decreasing, strictly convex function defined on the set $\mathbb{R}^+$ of strictly positive variances. Again, the hypotheses that f be strictly convex and monotonically decreasing are not restrictive.

Strategies. A strategy of a worker consists in choosing a precision (variance) v and corresponding effort $f(v)$. Similarly to the binary-verifiable model, let $\varepsilon \in \mathbb{R}$ be a variance threshold. We call a strategy v truthful if $v < \varepsilon$.

Supervision. In order to produce an incentive towards precise work, workers can be evaluated by the supervisor, or by a worker in a higher level. If the worker produces estimate x, while the supervisor or upper-level worker produces estimate y, the worker is penalized using the loss

function, for a penalty constant $c > 0$:

$$\ell(x,y) = c(x-y)^2 \quad . \tag{3.1}$$

In the following sections we will propose one level and hierarchical supervision schemes that provide incentive to workers so that they play with the truthful strategy.

3.5 One Level Supervised Schemes

In this section we study "one-level", or flat, supervision schemes where workers are directly verified by the supervisor. To verify a worker $u \in U$, the supervisor examines a task $i \in \partial u$ assigned to the worker. The supervisor then imposes a loss l to the worker depending on their solution. To provide an incentive to workers U to play with a truthful strategy, the supervisor chooses a subset of tasks to examine. We indicate with p the probability that a randomly chosen worker has a task that is being verified. By selecting first a random subset of m workers, and then picking an item for each worker, the supervisor can ensure a probability at least $p = m/|U|$ of verifying a worker. The higher the probability p, the higher the influence of the supervisor on all workers. We show that the number of items the supervisor needs to evaluate grows linearly with the numbers of items. The result of this section is very similar to the one in the work by [33]. We consider a setting where the supervisor provides precise answers without making mistakes. Such an assumption simplifies the proofs of theorems in this section, without limiting the generality of the results, as we consider the most favorable setting for the instructor to provide incentives. Still, the number of items to evaluate grows linearly with

the total number of workers. We will relax this assumption in the future sections and consider settings when the supervisor makes mistakes too.

Binary-verifiable model. Theorem 1 establishes a lower bound on p for the binary-verifiable model so that workers have an incentive to be truthful. Note that $-f'(x) > 0$ as the effort function is monotonically decreasing.

Theorem 1. *If every worker is assigned k tasks, the penalty cost equals C, and the probability p of being verified by the supervisor satisfies the following inequality:*

$$p > \frac{(-f'(\varepsilon))k}{C}, \tag{3.2}$$

then workers minimize their loss by playing with a truthful strategy.

Proof. The expected loss L of a worker $u \in U$ consists of two components: the effort to perform k tasks, and the expected penalty due to the supervisor when the worker provides a wrong solution

$$L(e) = kf(e) + epC \ . \tag{3.3}$$

The expected loss $L(e)$ is a convex function of e as a sum of a decreasing strict convex and an increasing linear function, where the decreasing convex function is bounded from below by 0. Thus it has a global minimum e^* that satisfies the equality

$$kf'(e^*) + pC = 0 \Rightarrow p = \frac{-f'(e^*)k}{C} \ . \tag{3.4}$$

We substitute the probability p in inequality (3.2) with the expression of (3.4) and obtain

$$\frac{-f'(e^*)k}{C} > \frac{-f'(\varepsilon)k}{C} \Rightarrow f'(e^*) < f'(\varepsilon) \ . \tag{3.5}$$

Because function $f'(e)$ is increasing as the derivative of a strictly convex function [73], it follows from the inequality $f'(e) < f'(\varepsilon)$ that $e < \varepsilon$. Therefore, to minimize the loss, a worker chooses to play with an error probability e^* such that $e^* < \varepsilon$, thus being truthful. ∎

Quantitative model. For the quantitative model, the following theorem establishes a similar lower bound on the probability p of being verified by the supervisor.

Theorem 2. *If every worker is assigned k tasks, the penalty constant of loss (3.1) is c , and the probability p of being verified by the supervisor satisfies the following inequality*

$$p > \frac{(-f'(\varepsilon))k}{c}, \tag{3.6}$$

then workers minimize their loss by playing with a truthful strategy.

We omit the proof as it follows similar steps as in the poof of Theorem 1.

For both models, the number of workers $m = p|U|$ that the supervisor needs to examine grows linearly with the total number of workers. This limits the applicability of the flat approach to relatively small task sets and worker crowds. In the following section, we develop a hierarchical setting that overcomes this limitation.

3.6 Hierarchical Supervised Schemes

In this section we develop hierarchical schemes that require a fixed amount of work by the supervisor to provide an incentive to workers for doing diligent work, regardless of the total number of workers. We first consider the case without redundancy when no task is guaranteed multiple workers; the case with redundancy, when every task is guaranteed multiple workers, is studied in Section 3.7 later. We develop hierarchical incentive schemes for both the binary-verifiable model and the quantitative model. In these schemes all workers perform tasks; there are no special meta-review tasks. The tasks are assigned so that each worker shares at least one task with a worker one level above in the hierarchy. By comparing the answers of the workers on these shared tasks, the workers at upper levels effectively check the work of workers at lower hierarchical levels. We note that the workers do not know which tasks they share with other workers; all they need to know, as we will show, is their level in the hierarchy. We will show that an incentive to be truthful does not deteriorate as the depth of a hierarchy grows.

The scheme organizes workers into a *supervision tree* (see Definition 1). The internal nodes of the supervision tree represent workers; the leaves represent tasks. A parent node and a child node share one task; this shared item is used to evaluate the quality of the child node's review work. At the root of the tree is the supervisor who is truthful, that is, she has a small probability of mistakes.

Definition 1. *A supervision tree of depth L is a tree with tasks as leaves, workers as internal nodes, and the supervisor as root. The nodes are grouped into levels $l = 0, \ldots, L-1$, according to their depth; the leaves are the nodes at level $L-1$ (and are thus all at the same depth). In*

the tree, workers at level $L-2$ perform the tasks they are connected to. Every node at level $0 \leq l < L-2$ performs exactly one task in common with each of its children.

To construct a supervision tree of branching factor at most k, we proceed as follows. We place the tasks as leaves and the above level of workers with at most k tasks per worker. Once level l is built, we build level $l-1$ by enforcing a branching factor of at most k. For each node x at level l, let $y_1, \ldots, y_n$ be its children. For each child $y_1, \ldots, y_n$, we pick at random a task s_i performed by y_i, and we assign node x with the task of examining the set $\{s_1, \ldots, s_n\}$ of tasks. At the root of the tree, we place the supervisor, following the same method for assigning tasks, that is, we assign the supervisor with doing one task from each of his or her children nodes, picked at random. Figure 3.2 illustrates a supervision tree with branching factor 2 and depth 3.

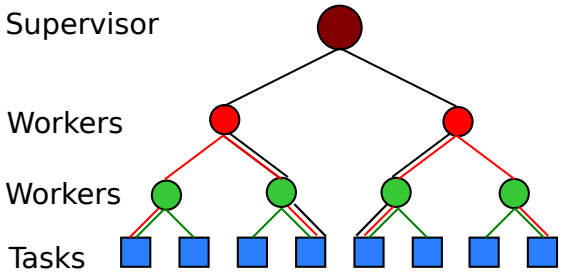


Figure 3.2: An example of a supervision tree with branching factor 2. The process starts bottom up. Each worker is assigned 2 tasks. For each depth-2 worker, a depth-1 worker is assigned one task in common with worker at depth-2 (red edges). The evaluation of the depth-2 worker will depend on the depth-1 worker. Similarly, the supervisor evaluates a depth-1 worker by reviewing one of the two tasks that the depth-1 worker has done(black edges).

In the following two subsections we study hierarchical schemes with binary-verifiable and quantitative tasks respectively.

3.6.1 The Binary-verifiable Model

We consider the case of a homogeneous worker population first, where workers have the same effort function. We find a tight bound on penalty cost C that ensures that a hierarchical scheme provides incentives. We then extend our results to the case of heterogeneous worker population, where worker effort functions are different, modeling a more realistic setting. The bound (3.11) will be crucial in distinguishing between workers who are proficient and workers with limited proficiency. We will show that if a population of workers is proficient on average, then a hierarchical scheme provides an incentive to be truthful.

Homogeneous worker population. We consider a setting where all workers have the same effort function f. We will show that the proposed hierarchical scheme provides incentives for worker to be truthful. First, to prove our main results, we formulate Lemma 1 that considers a worker and their superior in a supervision tree. It computes the expected loss of the worker, and it provides an upper bound on the worker's error probability.

Lemma 1. *Let workers $u, w \in U$ have error probabilities e_u, e_w respectively, and let worker w be the parent of worker u in a supervision tree with branching factor k and penalty cost $C > 0$. If worker u has effort function f then the expected loss $L(e_u, e_w)$ of worker u under the supervision of worker w is*

$$L(e_u, e_w) = kf(e_u) + e_u(1-e_w)C + (1-e_u)e_wC + e_ue_wD \ , \tag{3.7}$$

where D is a constant from the $[0, C]$ interval. Moreover, if there is $\sigma, \varepsilon \in (0, 1/2)$ such that

$e_w < \varepsilon$ *and*

$$C \geq \frac{f'(\sigma)k}{2\varepsilon - 1} \ , \tag{3.8}$$

then every $e_u^* \in \arg\min_{e_u} L(e_u, e_w)$ *satisfies inequality* $e_u^* < \sigma$.

Proof. The expected loss $L(e_u, e_w)$ of worker u consists of 4 components:

$$L(e_u, e_w) = kf(e_u) + (1 - e_u)e_w C + e_u(1 - e_w)C + e_u e_w D \ .$$

The first component $kf(e_u)$ is due to the effort of performing k tasks with error probability e_u. The other components account for the penalty that the superior w imposes in three mutually exclusive events. In particular, the second component $(1 - e_u)e_w C$ accounts for the event where worker u provides the correct solution to the common task but the superior makes a mistake. The third component $e_u(1 - e_w)C$ accounts for the case where the worker makes a mistake and the superior is correct. The fourth component $e_u e_w D$ accounts for the event where both worker and superior are incorrect; the penalty D belongs to the interval $[0, C]$, depending on the probability of the event when the worker and the superior have different answers and both of them are incorrect.

The loss $L(e_u, e_w)$ is a convex function of e_u as a combination of convex and linear functions. Therefore, the set $\arg\min L(e_u, e_w)$ is not empty. Let $e_u^* \in \arg\min L(e_u, e_w)$. Error

probability e_u^* satisfies the following inequality

$$kf'(e_u^*) - e_w C + (1-e_w)C + e_w D = 0 \Rightarrow f'(e_u^*) = \frac{(2e_w - 1)C - e_w D}{k} \ .$$

Combining it with the fact that $e_w, D \geq 0$, and with the assumption that $e_w < \varepsilon$, we obtain

$$f'(e_u^*) \leq \frac{(2e_w - 1)C}{k} < \frac{(2\varepsilon - 1)C}{k} \ . \tag{3.9}$$

Note that the $(2\varepsilon - 1)$ multiplier is negative as $\varepsilon < 1/2$. Because $(2\varepsilon - 1) < 0$, the right hand side of inequality (3.9) can be bounded using inequality (3.8)

$$\frac{(2\varepsilon - 1)C}{k} \leq \frac{(2\varepsilon - 1)}{k} \frac{f'(\sigma)k}{2\varepsilon - 1} = f'(\sigma) \ . \tag{3.10}$$

From inequalities (3.9, 3.10), it follows that $f'(e_u^*) < f'(\sigma)$. Because function f' is increasing as the derivative of a strictly convex function [73], we conclude that $e_u^* < \sigma$. ∎

The following theorem shows that in the proposed hierarchical scheme, we can choose the penalty C so that all Nash equilibria have the property that every worker plays truthfully.

Theorem 3. *Let the workers be organized into a supervision tree with branching factor k. If workers are rational and have the same effort function f, and penalty C satisfies the inequality*

$$C \geq \frac{f'(\varepsilon)k}{2\varepsilon - 1} \ , \tag{3.11}$$

for $\varepsilon \in (0, 1/2)$, then all Nash equilibria have the property that every worker in an equilibrium

plays truthfully.

Proof. The strategic choice of a player u depends only on the players placed above u. Thus the proof is by induction on the depth $l = 0, 1, \ldots, L-1$ of the tree. The inductive hypothesis is that the best response for players at depth up to l is to play with a truthful strategy, i.e. with error probability less that ε. At depth 0, the result holds trivially, as the supervisor plays a fixed truthful strategy.

To prove the inductive step, let us consider a worker u at level $l+1$ and the worker's superior w at level l, with error probabilities e_u and e_w respectively. The loss of worker u is solely depends on the superior w. By the inductive hypothesis, the superior w plays with a truthful strategy and therefore provides an incorrect solution to the common task with probability $e_w < \varepsilon$. The conditions of Lemma 1 for worker u with superior w are satisfied: superior w has error probability $e_w < \varepsilon$, and inequality (3.8) holds for $\sigma = \varepsilon$. Therefore, it follows from Lemma 1 that the best response of worker u is to play with error probability e_u^* that is less than ε, i.e. by choosing a truthful strategy. ∎

We can show that the inequality (3.11) is tight: if cost C does not satisfy inequality (3.11), then there exists an effort function, a supervision tree, and a level l such that a rational worker chooses a strategy with error probability greater than ε, that is, not a truthful strategy.

Theorem 4. *The bound proved in Theorem 3 is tight. Precisely, if the cost C does not satisfy inequality (3.11), then there exists an effort function, a supervision tree, and a level l such that a rational worker chooses a strategy with error probability greater than* ε.

Proof. To prove the theorem, we will show that when the cost C satisfies inequality

$$C < \frac{f'(\varepsilon)k}{2\varepsilon - 1} \ , \tag{3.12}$$

then there exists an effort function, a supervision tree, and a level l such that a rational worker chooses a strategy with error probability greater than ε. In particular, we choose effort function to be $f(x) = -\ln(x)$, and we assume that (3.12) holds for $\varepsilon \in (0, 1/4)$. To construct a tree we assume that the solution set A consists of 2 elements only. Let e_t be error probability of a worker on depth t of the supervision tree. We need to show that for large enough l a rational worker chooses $e_l > \varepsilon$. The superior on level 0 provides correct solutions, thus $e_t = 0$. The expected loss $L_t(e_t)$ of a worker u on level t is

$$L_t(e_t) = kf(e_t) + e_t(1 - e_{t-1})C + (1 - e_t)e_{t-1}C \ . \tag{3.13}$$

Worker u minimizes their loss by choosing e_t that sets the derivative of the loss to 0

$$kf'(e_t) + (1 - 2e_{t-1})C = 0 \Rightarrow f'(e_t) = \frac{(2e_{t-1} - 1)C}{k} \ .$$

Given that $f'(x) = -1/x$, the optimal value e_t is $\frac{k}{(1-2e_{t-1})C}$. The difference between e_t and e_{t-1} is

$$e_t - e_{t-1} = \frac{k/C}{1 - 2e_{t-1}} - e_{t-1} = \frac{k/C + 2e_{t-1}^2 - e_{t-1}}{1 - e_{t-1}} \ . \tag{3.14}$$

We are going to find a constant $\Delta > 0$ such that $e_t - e_{t-1} > \Delta$ for any $t \geq 0$. This would mean that as the tree depth increases, the probability of errors by workers would steadily increase, eventually surpassing the truthfulness threshold ε. To bound the right hand side of (3.14), we note that $1/(1-2e_{t-1}) \leq 1$ for $e_{t-1} \geq 0$. Therefore

$$e_t - e_{t-1} \geq k/C + 2e_{t-1}^2 - e_{t-1} \ . \tag{3.15}$$

Note that function $g(e_{t-1}) = 2e_{t-1}^2 - e_{t-1}$ is monotonically decreasing on the interval $[0, \varepsilon]$ as $\varepsilon < 1/4$. Assume that that all levels workers play with a truthful strategy, that is, $e_t < \varepsilon$ for any $t \geq 0$. We can further bound $e_t - e_{t-1}$ by using inequality $2e_{t-1}^2 - e_{t-1} > 2\varepsilon^2 - \varepsilon$ and inequality (3.15)

$$e_t - e_{t-1} > k/C + 2\varepsilon^2 - \varepsilon \ . \tag{3.16}$$

Inequality (3.12) implies that for some $\delta > 0$, $C = \frac{kf'(\varepsilon)}{2\varepsilon - 1} - \delta$. We use it to simplify the right hand side of (3.16)

$$e_t - e_{t-1} > k/C + 2\varepsilon^2 - \varepsilon = \frac{k}{\frac{k}{\varepsilon(1-2\varepsilon)} - \delta} - \varepsilon(1-2\varepsilon) \ .$$

For brevity, we denote $\varepsilon(1-2\varepsilon)$ as a.

$$e_t - e_{t-1} > \frac{k}{\frac{k}{a} - \delta} - a = \frac{ak}{k - a\delta} - a = \frac{a^2\delta}{k - a\delta} \ .$$

Expression $\frac{a^2\delta}{k-a\delta}$ is greater than 0 and does not depend on the level t of the hierarchy; we denote it as Δ.

$$e_t - e_{t-1} > \Delta \ . \tag{3.17}$$

The derivation of (3.17) is based on the assumption that $e_t < \varepsilon$. If we choose a hierarchy level l such that $l > \frac{\varepsilon}{\Delta}$, it follows from inequality (3.17) that $e_l - e_0 > \Delta * l > \varepsilon$. Because $e_0 = 0$, we conclude that $e_l > \varepsilon$ which contradicts to our assumption that $e_t < \varepsilon$ for $t \geq 0$. We have shown there exists a hierarchy level l such that $e_l > \varepsilon$ and the supervision tree does not provide incentive past depth l. ∎

Heterogeneous worker population. We now turn to the scenario when workers have different effort functions. We assume that workers in the supervision tree know their own effort function, but they do not know effort function of other workers. Thus, the strategic interactions between workers can be formulated as a game with incomplete information about effort functions. We introduce the definition of worker *proficiency* and we show that if the average proficiency of workers is sufficiently high, then all *interim* Bayes-Nash equilibria have the property that every proficient worker plays truthfully.

The proficiency of a worker is determined by the effort function of the worker, and it corresponds to the level of precision that is enforced by the supervision penalty to the worker. If the supervision penalty C is fixed, then the effort function f_u of a worker u determines whether the worker finds it worth achiveing the target precision ε or not.

Definition 2. *Given $\varepsilon \in (0, 1/2)$, penalty $C > 0$, and a worker $u \in U$ with effort function f_u, we*

say that the proficiency σ_u of u is the value x for which the following equality holds:

$$C = \frac{f'_u(x)k}{2\varepsilon - 1} \tag{3.18}$$

Further, we say that the worker is proficient if $\sigma_u \leq \varepsilon$.

Under natural assumptions on effort functions, equation (3.18) always has a unique solution. First, we assume that the more one tries to approach a zero probability of mistake, the more effort one needs to make, while having precisely zero probability would require infinite effort. This assumption translates into the following conditions: $\lim_{e\to+0} f_u(e) = +\infty$ and $\lim_{e\to+0} f'_u(e) = -\infty$. Secondly, random guessing requires no effort: $\lim_{e\to1/2} f_u(e) = 0$ and $\lim_{e\to1/2} f'_u(e) = 0$. The two natural assumptions guarantee that the derivative of f_u lies in the $(-\infty, 0)$ range. Moreover, the differentiablity and the convexity of f_u implies the continuity of f'_u [73], therefore, the continuity of f'_u and the strict convexity of f_u guarantees that the derivative takes any value in that range only at one point, that is, there are no two points with the same derivative. Thus, equation (3.18) always has a unique solution.

Definition 2 is justified by the following observation. Based on Lemma 1, value σ_u is the upper bound on the best response e^*_u of worker u under a supervision of a truthful worker. Therefore, error probability e^*_u of a *proficient* worker satisfies inequality $e^*_u < \sigma_u \leq \varepsilon$. This means that a *proficient* worker is truthful under a truthful superior. On the other hand, if a worker has limited proficiency, i.e. $\sigma_u > \varepsilon$, then, as we will show, the opposite of inequality (3.11) holds; and in this case, according to Theorem 4, a worker with that effort function in a supervision tree is not guaranteed to be truthful. We obtain the opposite of inequality (3.11)

from inequality $\sigma_u > \varepsilon$ by using the fact that f'_u is an increasing function as a derivative of a strictly convex function and the assumption that $\varepsilon < 1/2$:

$$\sigma_u > \varepsilon \Rightarrow f'_u(\sigma_u) > f'_u(\varepsilon) \Rightarrow \frac{f'_u(\sigma_u)k}{2\varepsilon - 1} < \frac{f'_u(\varepsilon)k}{2\varepsilon - 1} \Rightarrow C < \frac{f'_u(\varepsilon)k}{2\varepsilon - 1} \quad .$$

To reason about the diverse proficiency of workers in a population, we assume that the worker effort functions are distributed according to a probability distribution $\mathcal{F}$. We call a population of workers proficient as a whole if the following inequality holds:

$$\mathrm{E}_{f\sim\mathcal{F}}[\sigma_u] \leq \varepsilon \quad . \tag{3.19}$$

where σ_u is the proficiency of a random worker.

Note that a successful incentive scheme cannot guarantee that workers with limited proficiency have an incentive to be truthful, because it might take too much effort for such workers to have error probability smaller than ε. The following theorem shows that all *interim* Bayes-Nash equilibria have the property that every proficient worker plays with a truthful strategy.

Theorem 5. *Let workers be organized into a supervision tree with branching factor k and penalty $C > 0$. Let $\varepsilon \in (0, 1/2)$ and let $\mathcal{F}$ be a distribution of worker effort functions such that the population of workers is proficient as a whole, i.e. inequality (3.19) holds. If workers are rational, then all interim Bayes-Nash equilibria have the property that every proficient worker plays truthfully.*

Proof. In a supervision tree, the strategic choice of a player u depends only on the players placed above u. We prove by induction of depth $l = 1,\ldots,L-1$ of the tree that the best response of worker u is to play with error probability e_u^* such that $e_u^* < \sigma_u$, where σ_u is the solution of equation (3.18) with the worker's effort function f_u. Thus, by the definition of a proficient worker, it will follow that the best response of a proficient worker u is to play with $e_u^* < \sigma_u \leq \varepsilon$, i.e. to play truthfully.

At level 1, a worker u is evaluated by the supervisor who has error probability $e < \varepsilon$. Conditions of Lemma 1 for worker u are satisfied with $f = f_u$ and $\sigma = \sigma_u$. Thus, the best response of worker u is to play with $e_u^* < \sigma_u$.

To prove the inductive step, let us consider a worker u at level $l+1$ and her superior w at level l, with error probabilities e_u and e_w^* respectively. By the inductive assumption, $e_w^* < \sigma_w$. According to the first part of Lemma 1, the superior w induces loss $L(e_u, e_w^*)$ to the worker u that is defined by equation (3.7). However, worker u does not know the effort function of the superior. Thus, the loss of worker u is computed as the expectation of $L(e_u, e_w^*)$ over different effort functions of the superior

$$\begin{aligned}
\mathrm{E}_{f_w\sim\mathcal{F}} L(e_u, e_w^*) &= \mathrm{E}_{f_w\sim\mathcal{F}}[kf(e_u) + (1-e_u)e_w^* C + e_u(1-e_w^*)C + e_u e_w^* D] \\
&= kf(e_u) + (1-e_u)\mathrm{E}_{f_w\sim\mathcal{F}}[e_w^*]C + e_u(1-\mathrm{E}_{f_w\sim\mathcal{F}}[e_w^*])C + e_u\mathrm{E}_{f_w\sim\mathcal{F}}[e_w^*]D \\
&= L(e_u, \mathrm{E}_{f_w\sim\mathcal{F}}[e_w^*]) \ .
\end{aligned}$$

We use e^* to denote $\mathrm{E}_{f_w\sim\mathcal{F}}[e_w^*]$. Minimizing the loss $\mathrm{E}_{f_w\sim\mathcal{F}} L(e_u, e_w^*)$ with respect to e_u is equivalent to minimizing the loss $L(e_u, e^*)$. Therefore, the strategic choice e_u^* of worker u in the

presence of uncertainty about the type of the superior is equivalent to the strategic choice of worker u with a superior that has error probability e^*. We show that $e^* < \varepsilon$ by using the inductive assumption $e^*_w < \sigma_w$ and assumption (3.19), that the workers are proficient on average:

$$e^* = \mathrm{E}_{f_w \sim \mathcal{F}}[e^*_w] < \mathrm{E}_{f_w \sim \mathcal{F}}[\sigma_w] \leq \varepsilon \ .$$

The conditions of Lemma 1 are satisfied for the worker u with $f = f_u$, $\sigma = \sigma_u$, and $e_w = e^*$. It follows from Lemma 1 that the best response of worker u is $e^*_u < \sigma_u$. This finishes the inductive step and the proof of the theorem. ∎

What information do workers need? The schemes considered in this section organize workers into hierarchies. What information do workers need to know about the hierarchy, as they set to do their work? Do they need to be given the precise hierarchical scheme, including the names (or identities) of their supervisors? Or can they just be told that a hierarchy exists, without being told even what their place in it is? The interest in these questions lies in the fact that revealing to workers the identity of those above and below them in the hierarchy could create incentives to communicate via secondary channels and sway the outcome.

It turns out that the answer is somewhere in between: while workers do not need to know the identities of the workers above and below them in the hierarchy, they do need to know the level in which they are. The following pair of theorems makes this observation precise.

We denote the pure defection strategy where a worker always reports the same solution for *any* tasks as ξ.

Theorem 6. *Assume workers are organized into a supervision tree but they are not told their*

level in the tree. Then, for each $\varepsilon > 0$*, there are supervision trees where defecting with a constant strategy is a Nash equilibrium with a loss smaller than any truthful strategy.*

Proof. Let N and k be the number of players and the tree branching factor respectively. We analyze strategic choices of a worker $u \in U$ when all other workers $U \setminus u$ defect and play with strategy ξ. The worker can play a mix of the following two pure strategies. One strategy consists in playing the fixed move. This strategy carries a cost when the worker picks the wrong outcome and is reviewed by the supervisor; this happens with probability k/N. Thus, the expected cost of this strategy is bound by kC/N. The other strategy consists in playing an outcome that differs from the constant being played by defectors. Even leaving aside the cost of finding out the truth, this strategy carries a cost $(N-k)C/N$. So when $(N-k)C/N > kC/N$, or $N > 2k$, it is convenient to defect. The result is intuitive: it is convenient to defect when the probability of being reviewed by another defector is larger than the probability of being reviewed by the single supervisor. ∎

The following theorem essentially says that telling workers their level in the hierarchy is the minimum and sufficient amount of information required to ensure that collaborating is the only Nash equilibrium.

Theorem 7. *If there is a fixed upper bound k to the number of tasks that a worker is assigned, then the smallest amount of information a worker needs to know about the hierarchy to have an incentive to play with the truthfully strategy is* $\Theta(log \log N)$*, where N is the number of players in the hierarchy, and* $\Theta()$ *is the big-Theta notation of complexity theory.*

Proof. If we can give workers $\Theta(\log \log N)$ information or more, then we can tell them their

level in the hierarchy, and the induction argument in Theorem 3 applies.

Conversely, assume that we give fewer than $\Theta(\log\log N)$ bits of information to workers, and consider the situation for $N \to \infty$. The bits given out would induce a partition $C_1, C_2, \ldots, C_m$ of the workers, where workers receiving the same bits would belong to the same class. Assume that the partition classes are sorted according to size, so that $|C_1| < |C_2| < \cdots < |C_m|$. As the number of bits is smaller than $\Theta(\log\log N)$, for every $\gamma > 0$, there are n and j so that $|C_j| < \gamma |C_{j+1}|$. In other words, as the number of classes is less than logarithmic in N, as N grows, there must be arbitrarily large gaps in the ratios between sizes of adjacent classes. This implies that, for workers in C_{j+1} as above, the probability of being reviewed by a worker in levels $C_1 \cup \cdots \cup C_j$ can become arbitrarily small, since those workers can check on at most $k^2|C_1 \cup \cdots \cup C_j|$ workers below them. Thus, defecting becomes the preferred strategy by some of the workers if fewer than $\Theta(\log\log N)$ bits are communicated to the workers. ∎

3.6.2 The Quantitative Model

In the previous section we considered hierarchical schemes in the binary-verifiable setting. Workers report either a correct or incorrect answer in a task. We showed that hierarchical schemes provide incentives to be truthful in homogeneous worker populations if the penalty cost C is large enough. We also extended the result to the case of heterogeneous worker populations where workers have different levels of proficiency. In this section we study a quantitative setting in which workers can give a real number as an answer to a task. We will show that hierarchical schemes provide incentives to be truthful and that the strength of the incentive does not deteriorate with the depth of the hierarchy.

Let us consider a worker u who assigns value x to a task with answer $t \in \mathbb{R}$. Evaluation x is a random variable with variance $\sigma^2 = \mathrm{E}(x - \mathrm{E}[x])^2$ and bias $b = \mathrm{E}[x] - t$. The expected error v of the evaluation is $\mathrm{E}(x-t)^2$; this expected error can be represented as a sum of the variance σ^2 and squared bias b^2. Indeed, by adding and subtracting $\mathrm{E}[x]$ within $\mathrm{E}[(x-t)^2]$, expanding the squared sum and using the fact that $\mathrm{E}[x - \mathrm{E}[x]] = 0$, we obtain

$$\mathrm{E}[(x-t)^2] = \mathrm{E}[(x - \mathrm{E}[x] + \mathrm{E}[x] - t)^2] = \mathrm{E}[(x - \mathrm{E}[x])^2] + (\mathrm{E}[x] - t)^2 = \sigma^2 + b^2 \quad .$$

The following proposition specifies the expected penalty that a worker u with variance σ_u^2 and bias b_u receives when evaluated by a superior w with variance σ_w^2 and bias b_w.

Proposition 1. *Let workers $u, w \in U$ have variances σ_u^2, σ_w^2 and biases b_u, b_w respectively. If worker w supervises worker u by assigning a penalty according to function (3.1) with penalty constant $c > 0$, then the expected penalty $l(\sigma_u, b_u, \sigma_w, b_w)$ of worker u is*

$$l(\sigma_u, b_u, \sigma_w, b_w) = c(\sigma_u^2 + b_u^2 - 2b_u b_w + \sigma_w^2 + b_w^2) \quad . \tag{3.20}$$

Proof. We use E_u, E_w to denote the expectations over the evaluation of workers u and w respectively. Let x and y be evaluations to a task by workers u and w respectively. We simplify the expected loss $\mathrm{E}_u\mathrm{E}_w[c(x-y)^2]$ by replacing expression $(x-y)^2$ with the equivalent expression

$(x-t)^2 - 2(x-t)(y-t) + (y-t)^2$, and using the independence of evaluations x and y

$$l(\sigma_u, b_u, \sigma_w, b_w) = c\left(\mathrm{E}_u[(x-t)^2] - 2(\mathrm{E}_u[x]-t)(\mathrm{E}_w[y]-t) + \mathrm{E}_w[(y-t)^2]\right)$$
$$= c(\sigma_u^2 + b_u^2 - 2b_u b_w + \sigma_w^2 + b_w^2) \ .$$

∎

A worker u has control over the expected estimation error v that is the sum of σ^2 and b^2. To achieve the expected error v, worker u makes effort $f_u(v)$, where f_u is a strictly convex and decreasing function defined on $\mathbb{R}^+$.

Let workers be organized into a supervision tree with branching factor k, and let worker w be the parent of worker u. The expected cost of worker u is the sum of two components: the cost of performing k tasks, and the penalty due to the supervision by worker w. It directly follows from Proposition 1 that if superior w is unbiased, i.e. $b_w = 0$, then the best response has error v_u^* of worker u is

$$v_u^* = \arg\min_v \left(k f_u(v) + cv\right) \ . \tag{3.21}$$

Surprisingly, the best response of a worker to an unbiased superior does not depend on the precision of the worker's superior. This fact, which follows from additivity of variances of uncorrelated random variables, allows us to reason about the best response of a worker to an unbiased supervision without specifying the particular superior worker.

We adopt the following natural assumption on the population of workers. We assume

that the average bias of the best response to an unbiased supervision is 0. This assumption does not restrict individual workers to be unbiased.

Effort functions determine workers proficiency, as supervision penalty (3.1) can provide incentives to be truthful only if the cost of performing a task does not outweigh the penalty.

Definition 3. *Given* $\varepsilon > 0$ *and penalty constant* $c > 0$ *of loss (3.1), a worker* u *with effort function* f_u *is* proficient *if the best response to an unbiased supervisor has error* v_u^* *less than* ε.

Because workers do not know each other's effort functions, the strategic interaction of workers in a supervision tree can be formulated as a game with incomplete information.

The following theorem shows that workers in a supervision tree have incentives to be truthful.

Theorem 8. *Let rational workers be organized into a supervision tree with branching factor* k *and loss function (3.1). Let the population of workers have the property that the average bias of the best response to an unbiased supervision is 0. Then, all interim Bayes-Nash equilibria have the property that every proficient worker plays truthfully.*

Proof. We prove by induction of depth $l = 1, \ldots, L-1$ of the tree that the best response of a worker u is to play with v_u^* that is the solution of optimization problem (3.21). Thus, by the definition of a proficient worker, it will follow that the best response of a proficient worker u is to be truthful. At level 1, a worker u is evaluated by the supervisor who has bias 0, therefore worker u plays with the expected error v_u^*.

To prove the inductive step, let us consider a worker u at level $l+1$ and their superior w at level l. According to the inductive assumption, worker w plays with v_w^* that is the best re-

sponse to an unbiased supervision. Let b^*_w be a bias of worker w. According to Proposition 3.20, superior w induces penalty $c(v_u - 2b_u b^*_w + v^*_w)$ to worker u. Loss L of worker u under superior w is

$$L = kf_u(v_u) + cv_u - 2cb_u b^* w + v^*_w \ .$$

The expected loss L across different types of superiors is

$$\mathrm{E}[L] = kf_u(v_u) + cv_u - 2cb_u \mathrm{E}[b^*_w] + \mathrm{E}[v^*_w] \ .$$

Due to the assumption on the population of workers, we have $\mathrm{E}[b^*_w] = 0$. Therefore, the best response of worker u is to minimize $(kf_u(v_u) + cv_u)$, i.e. to play with the error probability v^*_u (Equation 3.21). This finishes the inductive step and the proof of the theorem. ∎

3.7 Incentives Schemes with Multiple Reviews per Item

In the incentive schemes proposed in the previous sections, many tasks will have only one worker assigned to it. In this section, we consider the case of crowdsourcing with redundancy, i.e., when each tasks has multiple workers assigned to it. This can be useful when it is possible to aggregate the answers produced by the workers into a single, higher accuracy answer.

3.7.1 One Level Supervised Schemes

When a task is performed by multiple workers, verifying a single task i can be used to verify all the workers in ∂i. The supervisor can leverage this in order to try to minimize the number of tasks to be verified, while guaranteeing a worker verification probability p that satisfies Theorem 1. We will show that when a graph G of tasks and workers is given, i.e. when we do not have control over task allocation, then constructing the smallest subset S is $\mathcal{NP}$-hard, and it can be proved by reduction from vertex-cover. However, if we can control tasks allocation, then we can easily construct graphs on which the set of tasks that need verification is as small as possible.

The assignment graph is given. We first study a scenario in which the worker-task assignment is fixed, and we must choose the subset $S \subseteq I$ of tasks verified by the supervisor. When the supervisor examines a task $i \in I$, she evaluates all the workers ∂i who were assigned to the task i. Figure 3.3a illustrates a case when examining 3 tasks is enough to evaluate all the workers.

The supervisor wants to spend the least amount of effort to evaluate at least m workers. For the case $p = 1$, or $m = |U|$, the supervisor needs to find the smallest subset of tasks such that every worker is assigned one task from the subset. We name the problem of finding such a set the *Superior Assignment* problem (abbreviated to *SA*). The following theorem shows that the *Supervisor Assignment* Problem is $\mathcal{NP}$-hard.

Theorem 9. *Supervisor Assignment problem is $\mathcal{NP}$-hard.*

Proof. We will show that finding the smallest vertex cover for any graph is an instance of the Supervisor Assignment problem. Thus, solving the *Supervisor Assignment* problem is at least

as hard as solving Vertex Cover.

Let $G = (V, E)$ be an arbitrary graph with vertexes V and edges E. We construct a bipartite revision graph G' for a set of workers U and set of items I by taking $U = E$ and $I = V$: that is, we use workers in our bipartite graph to represent the edges of the original graph. Each worker $u \in U$ is assigned to review items v_1, v_2, where in G the edge u connects v_1 and v_2. The graph G' is called the incidence graph [39]. It is immediate to see that a subset of vertices $V' \subseteq V$ is a *vertex cover* for G if and only if picking all items in V' enables the verification of all workers E of G'. Thus, Vertex Cover can be reduced to the Supervisor Assignment problem. ∎

We now show that, if every worker is assigned at most k tasks, there are fast k-approximation algorithm for SA. A k-approximation algorithm finds a subset S' of tasks such that $|S'| < k|S|$, where S is the optimal solution.

We will show that the SA problem on graph G is equivalent to the VC problem on a hypergraph with edge size at most k. A hypergraph $H = (V, F)$ is a set of vertices V and hyperedges F. A hyperedge $f \in F$ connects a subset of edges from V. Hypergraph H has edge size at most k if every edge $f \in F$ contains at most k nodes. There are known simple k-approximation algorithms for VC on k-bounded hypergraphs [41].

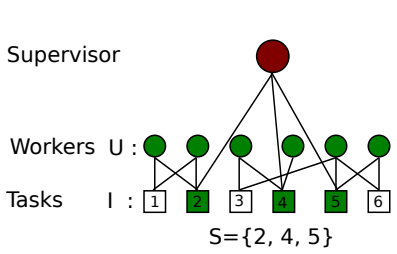
Proposition 2. *The Supervisor Assignment problem for a bipartite review graph $G = (U \cup I, E)$ with degree at most k is equivalent to Vertex Cover for a hypergraph with edge size at most k.*

Proof. The SA problem is immediately equivalent to a VC cover for a hypergraph that has U as vertex set, and has I as edge set, where each edge $i \in I$ connects the vertices that correspond to the workers to which i is assigned. ∎

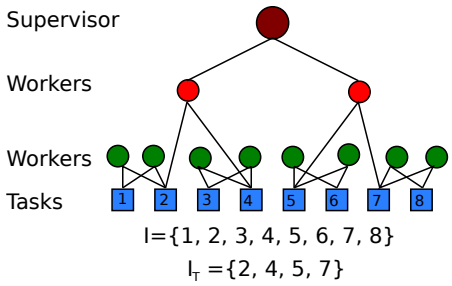
A simple k-approximation algorithm works as follows. Let $G = (V \cup F, E)$ be a bipartite graph and $S = \emptyset$. While set E is not empty, we randomly choose an edge $(w, f) \in E$, add node w to set S, and delete all edges incident to w or f. When E is empty, set S is a k-approximation to the SA problem on graph G.

The assignment graph can be constructed. If we can construct the assignment graph, then it is easy to ensure optimality. When all workers have only one task in common, the supervisor can evaluate all the workers by verifying only one task. From a crowdsourcing perspective, however, concentrating effort of all workers on one assignment has the unwelcome effect that all other tasks receive fewer workers. If we use worker multiplicity for a task in order to achieve higher reliability in the solution of a task, this is undesirable. A natural assumption is to require the review graph G to be k-regular.

To construct a k-regular review graph, we proceed as follows. We select $n = \lceil \frac{|U|}{k} \rceil$ "peg" tasks first. Each of these peg tasks will be done by a set of k non-overlapping workers, so by verifying the n peg tasks, the supervisor is able to verify all workers ($p = 1$). For smaller values of the verification probability p, the supervisor can simply choose to verify a randomly chosen subset of the peg tasks. We assume, of course, that the workers cannot compare their work with each other, so that they cannot infer which tasks are the peg tasks among those they are assigned. Once the peg tasks and their reviewers are chosen, we assign the other tasks to workers in any way that leads to k-regularity. It is easy to see that this construction is optimal, for $|U|$ workers doing k tasks each cannot be verified by picking fewer than n items.



(a) An example of a graph with all 6 workers being evaluated based on a set of 3 tasks. The supervisor inspects tasks 2, 4 and 5 that connected to all workers. The tasks and workers the supervisor reaches out are colored.



(b) A supervision hierarchy that is a union of a supervision tree and a bipartite graph of workers and tasks. Every task is assigned to at least 2 workers. The set of tasks I_T in the tree is a subset of tasks I in the bipartite graph. Every worker is assigned at least one tasks from the set I_T.

Figure 3.3: A bipartite graph evaluated by the supervisor, and a supervision hierarchy.

3.7.2 Hierarchical Supervised Schemes

A *supervision hierarchy* combines a bipartite graph of workers and tasks and a supervision tree. The supervision tree provides an incentive while the bipartite graph ensures that every task is assigned to several workers.

Definition 4. *A supervision hierarchy is a connected graph that consists of two subgraphs: a bipartite graph $G = (U \cup I, E)$ and a supervision tree T with workers U_T and tasks I_T. The set of tasks I_T is a subset of tasks I and for every worker $u \in U$ there is a task $i \in I_T$ such that the edge (u,i) belongs to E.*

Figure 3.3b illustrates such a supervision hierarchy. The supervisor provides an incentive for the two immediate subordinate workers while these workers provide the incentive to the rest of workers by performing a total of 4 tasks.

For a given bipartite graph G the task of constructing the smallest supervision hierar-

chy is $\mathcal{NP}$-hard. Indeed, the subset I_T of I has the property that every worker $u \in U$ has at least one task from I_T. Thus finding the smallest set I_T is an instance of the *Supervision Assignment* problem we discussed in the previous section; and showed that it is an $\mathcal{NP}$-hard problem.

Theorems 5, 8 can be extended to show that supervision hierarchies provide incentives to be truthful for the binary-verifiable and the quantitative settings respectively. Indeed, the theorems hold directly for workers that belong to the tree T of the supervision hierarchy. On the other hand, every worker in graph G of the supervision hierarchy has a superior worker from the bottom level of tree T. Thus, workers in G can be considered as one extra layer of workers within tree T.

3.8 Conclusions

We proposed and analyzed supervision incentive schemes that ensure that the optimal strategy for workers is to be truthful. The schemes rely on hierarchies in order to scale to arbitrarily large sets of items and workers, while requiring only a constant amount of work on the part of the supervisor. In the hierarchy, workers are organized in layers, and every layer exerts an incentive over the layer below, ensuring that the optimal behavior of the workers is sufficiently precise. We show that the truthful incentive holds even in populations of workers with diverse proficiency, where workers can have limited proficiency, provided that there are enough proficient workers in the crowd. Interestingly, the only information the workers need to know about the hierarchy is their level in it: they do not need to know the identities of their supervisors or subordinates, nor which tasks they share, and all workers perform exactly

the same work. In particular, there are not two flavors of "normal" and "metareview" tasks. Our schemes graciously extend from binary verifiable tasks to quantitative tasks making them relevant to a wide range of crowdsourcing applications.

Chapter 4

Authorship Tracking for Revisioned Content

4.1 Problem Setting and Contributions

Versioned content is abundant on the Web. Wikipedia, and wikis, constitute the most prominent example, and they account for a large portion of total page-views. Blogs with multiple authors, and pages served by content-management systems, are another example in which the versioning is present, but not directly exposed to the viewer. Code is another prominent example of revisioned content, and one that is becoming common on the web, thanks to the success of sites like GitHub, where users can share their code repositories.

We study in this paper the problem of attributing revisioned content to its author, and more generally, to the revision where it was originally introduced. This problem is insteresting for several reasons. The Wikipedia Reuse Policy[1] requires people reusing Wikipedia material to either provide a link to the original article and revision history, or to cite the most prominent authors of the content. Furthermore, in the Wikipedia community it is felt that proper content

[1] http://en.wikipedia.org/wiki/Wikipedia:Reusing_Wikipedia_content

attribution is an important way to acknowledge and reward contributors, and to foster participation and contributions from communities where authorship has been traditionally recognized and rewarded, such as the academic community [30, 61]. Tracking the authorship of Wikipedia content is also an important tool in assisting editors, and viewers, in determining the origin of assertions, and analyzing page evolution. In code, as in wikis, authorship tracking is useful to properly reward contributors. Furthermore, authorship tracking can be useful in determining the reason behind implementation choices. Several revisioning systems implement "blame" methods, which attribute every line to an author/revision, but this attribution is extremely crude and imprecise, as it cannot cope with blocks of code that are transposed from one location to another, or from one file to another — changes that are common when code is polished or refactored.

At first glance, the attribution problem for revisioned content seems trivial: surely we can simply compare each revision with the previous one, detect any new content, and attribute it to the revision's author. Unfortunately, things are not quite so simple. Content in revisioned systems is often deleted, only to be later introduced, and it is important to be able to trace the authorship to the first original introduction. In Wikipedia, the content of pages is frequently removed by vandals, and re-instated in subsequent revisions: this is illustrated in Figure 4.10, where the periodic dips in page size correspond to content deletions. One way to guard against such attacks is to check whether the most recent revision happens to coincide with one of the previous revisions, in which case, authorship is carried over from the previous revision. However, this ad-hoc remedy cannot cope with broader attacks. For instance, attackers could first use a fake identity to remove the page contents, then use their main identity to restore the page to its previous contents, except for some small, imperceptible changes that foil the revision

equality check: the whole page content would then be attributed to them. The goal of this work is to present algorithms that can be used on Wikipedia, with the resulting authorship information available to visitors. Once authorship information is prominently displayed, attacks that aim at inflating the size of one's authorship are likely, prompting our quest for general, robust algorithms. A more general solution also benefits code attribution, since blocks of code are commonly moved from one branch to another, or deleted and later re-inserted.

We propose to attribute authorship of revisioned content by comparing the content of the most recent revision, with the entire content of all previous revisions. For every symbol (word, or character, or token) in the most recent revision, we compute all statistically significant matches with previous content: these are the matches whose sequence of symbols is rare enough that the match is likely to be due to a shared origin, rather than serendipitous re-invention. The symbol is then assigned the earliest possible origin that is compatible with all the matches. We call this approach the *earliest plausible attribution* approach. We show that earliest plausible attribution yields a more natural content attribution than other approaches, including approaches based on longest matches with previous content, or approaches inspired by the edit-analysis work of Tichy [76]. By comparing new revisions with the full set of previous revisions, the earliest plausible attribution approach achieves resistance to page deletions and vandalism. As Figure 4.9 (A0 vs. A1) later in the paper indicates, the resulting attribution differs by over 75% from the attribution computed via comparisons to the most recent revision only, the difference being due chiefly to deletion-reinsertion attacks and other vandalism.

We introduce efficient algorithms for earliest plausible attribution. If fed all revisions at once, the algorithms can compute the content origin in time proportional to the size of the

revision history, which is clearly optimal. More commonly, though, revisions are created and must be analyzed one and a time. A practical implementation must maintain a summary of all past revisions, and process a new revision on the basis of such a summary. We show that the algorithm we propose uses a summary of size proportional to all the past *change* in the previous revisions — and this change size is typically much smaller than the total revision history size, since a new revision is usually identical to the preceding one except for a few small changes. The algorithm runs in time proportional to the sum of the size of the previous summary, and the size of the most recent revision. Again, since both summary and most recent revision must be read, this is optimal.

4.2 Related work

The WikiTrust tool computes a value of reputation for Wikipedia authors and content, as well as the revision where each word was inserted [2, 3, 1]. The attribution algorithm achieves resistance to vandalism by comparing the most recent revision not only with the preceding one, but also with a set of "reference" revisions, consisting of recent revisions that either have high content reputation, or that were created by a high-reputation author. The approach is fairly effective in practice, but the attribution depends on the reputation computation: there is no independent characterization of the attribution that is computed, and the process is computationally involved. Furthermore, it is not clear how to extend the approach beyond Wikipedia.

In [1], several text matching algorithms are evaluated for their ability to explain the editing process in Wikipedia. Tichy-inspired algorithms [76] were found to be highly efficient,

and as precise as any alternative, for the problem of comparing two revisions. In contrast, in this work we show that for the problem of comparing a revision with all the preceding ones, the earliest plausible attribution yields more efficient algorithms, and arguably more natural results. The problem of attributing Wikipedia content has also been studied in [29], where an algorithm based on hierarchical content matching is proposed. When a revision is compared with the preceding revision, matches of large sections of text (sections, paragraphs, sentences) are evaluated first, and a finer-grained algorithm based on the Python difflib library is used to attribute any remaining content. The resulting attribution is reported to compare favorably with the one computed by WikiTrust.

String matching is a very well studied problem; see e.g. [40] for an in-depth overview. The algorithms presented in this paper make use of several results on string matching, including tries and suffix trees. Sophisticated string matching algorithms developed for genetic applications involve a two-step process: a coarse alignment is computed between the strings, followed by a finer-grained analysis of string differences (see [40] again). This "genetic" approach is resistant to the transcription errors that occur in gene sequencing. The algorithms developed in this paper are based instead on exact matching of short sequences. It is an interesting open question whether the algorithms for attribution of revisioned content could benefit from the genetic approach.

The attribution problem considered in this paper is a special case of *information provenance* problem. For an overview of information provenance, see e.g. [10, 11] for provenance in databases, and [74, 59, 32] for an overview in a broader context.

$\mathcal{A}$

Paper organization. After introducing some notation and concepts, we compare in Section 4.4 conceptual methods of defining attribution, providing justifications for our choice of earliest plausible attribution. In 4.5 we describe an efficient algorithm for earliest plausible attribution, and we prove that the algorithm is optimal. We present empirical results obtained in the analysis of the English Wikipedia in Section 4.6, and we conclude with some discussion of the results and possible future work in Section 4.7. All the code and data for the algorithms can be found at `https://sites.google.com/a/ucsc.edu/luca/the-wikipedia-authorship-project`.

4.3 Definitions

Revisions. We model revisioned content as a sequence of revisions $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$. Each revision ρ consists in a sequence of tokens $t_0, t_1, \ldots, t_{m-1}$, taken from a set $\mathcal{T}$ of tokens, where $len(\rho) = m$ is the length of ρ. We assume that $len(\rho_0) = 0$, so that ρ_0 represents the initial, empty revision that exists before any subsequent revision is created. For $\rho = t_0, t_1, \ldots, t_{m-1}$, we indicate with $\rho[i]$ the token t_i, and we write $\rho[i:j]$ for $t_i, t_{i+1}, \ldots, t_{j-1}$. Depending on the application, the tokens can be individual unicode characters, or they can be words in a text, tokens of a programming language, and so forth. Given a sequence $\overline{\rho}$ of revisions, a *global position* is a pair (n,k) with $n \geq 0$ and $0 \leq k < len(\rho_k)$. Thus, a global position denotes a token occurrence at a particular revision. In a Wikipedia page, for instance, the global position (n,k) may denote the k-th word of the n-th revision.

Matches. A *match* $M = (n, i, j, n', i', j')$ between positions $[i..j-1]$ of revision ρ_n and positions $[i'..j'-1]$ of revision $\rho_{n'}$, denoted informally (and more intuitively) as $M = (\rho_n[i:$

$j] = \rho_{n'}[i' : j'])$, consists of two revisions ρ_n, $\rho_{n'}$, and indices $0 \leq i < j \leq len(\rho_n)$, $0 \leq i' < j' \leq len(\rho_{n'})$, such that:

- $j - i = j' - i' > 0$, so that the matched portions have equal length and are non-empty;
- for all $0 \leq k < j - i$, we have $\rho_n[i + k] = \rho_{n'}[i' + k]$, so that tokens at corresponding positions of ρ_n and $\rho_{n'}$ match.

Given a match $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$, we denote by $len(M) = j - i$ its length. We say that a position k is *matched* by M if $i \leq k < j$. For a position k matched by M, we let $M(n,k) = (n', k - i + i')$: thus, we think of matches as partial functions between global positions that relate positions filled by equal tokens. We denote by $\mathcal{M}(\rho_n, \rho_m)$ the set of all matches between revisions ρ_n and ρ_m. We say that a match $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$ is a *sub-match* of $M' = (\rho_n[\hat{\imath} : \hat{j}] = \rho_{n'}[\hat{\imath}' : \hat{j}'])$ if $\hat{\imath} \geq i$, $\hat{j} \leq j$, and $\hat{\imath} - i = \hat{\imath}' - i'$; we say that the sub-match is *proper* if at least one of the two inequalities is strict.

Interesting matches. Our interest in matches is due to the fact that a match between a later revision and earlier one may indicate that the content of the later revision originated in the earlier one. Not all matches correspond to a common origin of the content, however. For instance, in English, the two-word sequence "such that" is very common, and it would be unreasonable to assume that they have been copied from an earlier revision whenever they appear in a later one. In order to use matches to study authorship, we need to distinguish fortuitous matches from those that indicate shared origin. An in-depth approach would likely require a probabilistic model of content structure, and of how content propagates from one revision to the next. Such a model could then be used to compute, for each revision token, a

probability distribution over the places where the content might have originated.

We follow a simpler, discrete approach, where content is attributed deterministically to a revision of origin. Deterministic attribution leads to efficient algorithms that can scale to very large bodies of content, such as Wikipedia. We also remark that users of authorship information generally expect a deterministic attribution: Wikipedia visitors and editors want to know who wrote what, and copyright is based on deterministic, not probabilistic attribution. Probabilistic attribution algorithms, and the question of their efficient implementation at scale and possible accuracy advantages, remain a topic for future work.

Consider a match $M = (\rho_n[i:j] = \rho_k[i':j'])$ between two revisions ρ_n and ρ_k, with $k < n$. To decide whether to attribute the sequence $\sigma = \rho[i], \rho[i+1], \ldots, \rho[j-1]$ to ρ_n or ρ_k, we use a *rarity* function $\gamma : \mathcal{T}^* \mapsto \mathbb{R}^+$: intuitively, the larger $\gamma(\sigma)$ is, the more likely it is that the sequence σ in ρ_n and ρ_k shares the same origin. We require that a rarity function γ satisfies the following two conditions:

- $\gamma(\emptyset) = 0$: the rarity of the empty sequence is zero.
- For all $\sigma \in \mathcal{T}^*$ and all $t \in \mathcal{T}$, we have $\gamma(\sigma) < \gamma(\sigma t)$: longer sequence are strictly rarer than shorter ones.

A simple choice is $\gamma(\sigma) = len(\sigma)$: the rarity of a sequence is equal to its length. More sophisticated rarity functions can be used: for instance, if we know the occurrence probability p_t of each token t, we can take $\gamma(t_0, t_1, \ldots, t_m) = \prod_{i=0}^{m} \frac{1}{p_{t_i}}$. Rarity functions based on the occurrence frequency of multi-token sequences could also be used.

Given a match $M = (\rho_n[i:j] = \rho_k[i':j'])$, we define its interest $\gamma(M) = \gamma(\rho_n[i], \rho_n[i+$

$1], \ldots, \rho_n[j-1])$ to be equal to the rarity of the matched sequence of tokens. We define the *interesting matches between revisions* ρ_n *and* ρ_m*, according to the rarity function* γ *and threshold* Δ, as the set of matches of rarity at least Δ:

$$\mathcal{M}(\rho_n, \rho_m \mid \gamma \geq \Delta) = \{M \in \mathcal{M}(\rho_n, \rho_m) \mid \gamma(M) \geq \Delta\} \ .$$

We note that if we choose $\gamma = len$, the set $\mathcal{M}(\rho_n, \rho_m \mid len \geq l)$ will consist of all matches between ρ_n and ρ_m that have length at least l. Given a position $0 \leq k < len(\rho_n)$ of revision ρ_n, we denote by $\mathcal{M}[k](\rho_n, \rho_m \mid \gamma \geq \Delta)$ the interesting matches between ρ_n and ρ_m that have interest at least Δ as measured by γ, and that match position k of ρ_n.

Origin labeling. An origin labeling associates with each token the revision where the token originated. Precisely, an *origin labeling* Θ for a (finite or infinite) sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$ of revisions is a labeling that associates with each global position (n,k) of $\overline{\rho}$ its *origin* $\Theta(n,k) \in \mathbb{N}$, with $\Theta(n,k) \leq n$. If $\Theta(n,k) = n$, we say that the token $\rho_n[k]$ is new in ρ_n.

4.4 Conceptual Algorithms

In some instances of revisioned content, such as Google Docs, full information about the edit actions by each individual user are available. In this case, the authorship can be computed by observing directly the typing, cutting, pasting, etc, performed by each editor. In many other instances, however, we can observe only the outcome of the editing process, namely, the sequence of revisions produced by the various users. This is the case for Wikipedia, and for code repositories, since the environments where users edit the code are independent from the

repositories. In these cases, we must infer authorship after the fact, by comparing the result of the editing with previous content. There is no a-priori correct way to infer authorship, as we cannot reconstruct the mental process of the editors to tell whether they are copying or reinventing. One of the contributions of this paper is to introduce the notion of *earliest plausible attribution* for revisioned content, showing that it leads to plausible attribution in practice. We remark that, even when the actions of users are observable during editing, as in Google Docs, we can never be sure whether editors are retyping a passage, copying it from paper, or reinventing it anew: earliest plausible attribution can thus be a useful notion even when edit actions are observable in detail. In this section we define earliest plausible attribution and we compare it with other attribution methods. The question of efficient implementation will be the subject of the next section.

4.4.1 Comparison with preceding revision

Algorithm A0 computes the origin of tokens in a revision ρ_n by comparing the revision with the previous one in the sequence. Given a sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$ of revisions, with $\rho_0 = \emptyset$ as the initial empty revision, algorithm A0 computes an origin labeling Θ for $\overline{\rho}$ proceeding inductively on the revisions. The first revision ρ_0, being empty, has a null labeling. For each subsequent revision ρ_n, $n > 0$, algorithm A0 computes all interesting matches with the preceding revision ρ_{n-1}. Every unmatched token in ρ_n is assigned an origin label of n. Each matched token is assigned the origin label of the matching position in the previous revision; if the token had multiple matches to different positions, the token is assigned the minimum of the origin labels of the corresponding positions.

Algorithm A0 Matches with previous revision.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

1: **for** revisions $n = 1, 2, 3, \ldots$ **do**
2: **for all** positions $0 \leq k < len(\rho_n)$ of ρ_n **do**
3: Let $\widehat{\mathcal{M}} := \mathcal{M}[k](\rho_n, \rho_{n-1} \mid \gamma \geq \Delta)$.
4: **if** $\widehat{\mathcal{M}} = \emptyset$ **then**
5: $\Theta(n,k) := n$
6: **else**
7: $\Theta(n,k) := \min_{M \in \widehat{\mathcal{M}}} \Theta(M(n,k))$.
8: **end if**
9: **end for**
10: **end for**

One may conceive a variant algorithm, termed Algorithm A0M, where only the most interesting match(es) for each token are considered: the idea being that the longer the match, the more likely it is to correspond to origin. Algorithms A0 and A0M may yield different labelings, as illustrated in Figure 4.1. In the figure, we use sequence length as the rarity function, together with a threshold of 3, so that matches that are 3 or more tokens are considered interesting. In labeling symbols `b c` in ρ_3, Algorithm A0 considers two interesting matches: ($\rho_3[0:3] = \rho_2[0:43$), involving `a b c`, and ($\rho_3[1:6] = \rho_2[4:9]$), involving `b c z z z`. The first match yields origin `1 1` for `b c`, the second `2 2`. The origin assigned by A0 is the least of these two, namely, `1 1`. Algorithm A0M, on the other hand, considers only the second match, as it is longer, and assigns to `b c` origin `2 2`.

This example highlights why we prefer to consider all interesting matches, rather than just the longest ones: even though `a b c` in ρ_3 matches `a b c` in ρ_1, it is assigned origin `1 2 2` according to A0M. We take the point of view that a match that is interesting (with a matched sequence of tokens that is sufficiently rare) denotes a common origin of the content. If there is more than one interesting match for a token position, we look at all such interesting matches as possible explanations for the origin of the content, and we err on the side of the oldest possible attribution, yielding the min in line 7 of Algorithm A0.

Algorithm A0M Origin via most interesting matches with previous revision.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

1: **for** revisions $n = 1, 2, 3, \ldots$ **do**
2: **for all** positions $0 \leq k < len(\rho_n)$ of ρ_n **do**
3: Let $\widehat{\mathcal{M}} := \mathcal{M}[k](\rho_n, \rho_{n-1} \mid \gamma \geq \Delta)$.
4: **if** $\widehat{\mathcal{M}} := \emptyset$ **then**
5: $\Theta(n,k) = n$
6: **else**
7: Let $\widetilde{\mathcal{M}} = \arg\max_{M \in \widehat{\mathcal{M}}} \gamma(M)$.
8: $\Theta(n,k) := \min_{M \in \widetilde{\mathcal{M}}} \Theta(M(n,k))$.
9: **end if**
10: **end for**
11: **end for**

4.4.2 Earliest plausible attribution

Algorithm A0 (and A0M) relies on comparisons with the immediately preceding revision only. In many relevant examples of versioned content, content can be deleted from one revision only to reappear several revisions later. For instance, the content of Wikipedia pages is frequently deleted by vandals. If authorship is determined via a comparison with the immediately preceding revision only, then an editor who restores the contents of a Wikipedia page after it is deleted would be attributed the authorship of all the restored content. As these periodic acts of vandalism that destroy most of a page's content are common on Wikipedia, authorship algorithms that are based only on comparisons with the immediately preceding revision will grossly mis-attribute content, as we will show experimentally in Section 4.6.

Our preferred algorithm for attribution of revisioned content, Algorithm A1, compares the latest revision with *all the previous revisions,* looking for matches with any prior content, rather than just content in the immediately preceding revision. We call this process *earliest plausible attribution,* since the attribution it produces is the earliest that is compatible with an explanation by interesting matches. Figures 4.2 and 4.3 provides a comparison of algo-

ρ_3:	$a^1\ b^1\ c^1\ z^2\ z^2\ z^2$	ρ_3:	$a^1\ b^2\ c^2\ z^2\ z^2\ z^2$
ρ_2:	$a^1\ b^1\ c^1\ x^2\ b^2\ c^2\ z^2\ z^2\ z^2$	ρ_2:	$a^1\ b^1\ c^1\ x^2\ b^2\ c^2\ z^2\ z^2\ z^2$
ρ_1:	$a^1\ b^1\ c^1$	ρ_1:	$a^1\ b^1\ c^1$

(a) A0 (b) A0M

Figure 4.1: A sequence of revisions, with origin labeled according to algorithms A0 and A0M. We represent each revision by its list of tokens, using letters to denote tokens. The origin labels are computed for a rarity function equal to sequence length, and threshold of 3. We write above every token the origin that the algorithm assigns to it.

rithm A0 and A1 in presence of a delete-and-restore attack, as common on Wikipedia, and of a more complex attack involving content that is deleted, then gradually re-instated.

Algorithm A1 Origin via interesting matches with all preceding revisions.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

1: **for** revisions $n = 1, 2, 3, \ldots$ **do**
2: **for all** positions $0 \leq k < len(\rho_n)$ of ρ_n **do**
3: Let $\widehat{\mathcal{M}} := \bigcup_{0<m<n} \mathcal{M}[k](\rho_n, \rho_m \mid \gamma \geq \Delta)$.
4: **if** $\widehat{\mathcal{M}} = \emptyset$ **then**
5: $\Theta(n,k) := n$
6: **else**
7: $\Theta(n,k) := \min_{M \in \widehat{\mathcal{M}}} \Theta(M(n,k))$.
8: **end if**
9: **end for**
10: **end for**

4.4.3 Tichy-based matching

One of the better-known algorithms for generating edit differences between revisions is due to Tichy [76]. Since the Tichy algorithm performs well in explaining the edit history of Wikipedia [1], it is of interest to adapt it to origin computation and compare it to A1. Given a revision $\rho_n = t_0, t_1, \ldots, t_{m-1}$, the Tichy-based Algorithm A2 searches revisions $\rho_0, \ldots, \rho_{n-1}$ for the longest prefix of $t_0, t_1, \ldots, t_{m-1}$. If this longest prefix is, say, $t_0, t_1, \ldots, t_k$, with $k \leq m-1$ and $\gamma(t_0, t_1, \ldots, t_k) > \Delta$ for the chosen rarity function γ and threshold Δ, then the algorithm fixes the

ρ_4: a^4 b^4 c^4 x^4 f^4 g^4 h^4	ρ_4: a^1 b^1 c^1 x^2 f^1 g^1 h^1
ρ_3: p^3 q^3	ρ_3: p^3 q^3
ρ_2: a^1 b^1 c^1 x^2 f^1 g^1 h^1	ρ_2: a^1 b^1 c^1 x^2 f^1 g^1 h^1
ρ_1: a^1 b^1 c^1 f^1 g^1 h^1	ρ_1: a^1 b^1 c^1 f^1 g^1 h^1
(a) A0	(b) A1

Figure 4.2: A sequence of revisions, with origin labeled according to algorithms A0 and A1, with rarity equal to length and threshold 3. This sequence illustrates a delete-and-restore event, common on Wikipedia.

ρ_6: a^4 b^4 c^4 d^4 e^6 x^6 w^6 g^6 h^6 l^6	ρ_6: a^1 b^1 c^1 d^1 e^1 x^2 w^6 g^1 h^1 l^1
ρ_5: q^3 r^3 a^4 b^4 c^4 d^4 f^5 g^5	ρ_5: q^3 r^3 a^1 b^1 c^1 d^1 f^5 g^5
ρ_4: p^3 q^3 r^3 a^4 b^4 c^4 d^4	ρ_4: p^3 q^3 r^3 a^1 b^1 c^1 d^1
ρ_3: p^3 q^3 r^3	ρ_3: p^3 q^3 r^3
ρ_2: a^1 b^1 c^1 d^1 e^1 x^2 f^1 g^1 h^1 l^1	ρ_2: a^1 b^1 c^1 d^1 e^1 x^2 f^1 g^1 h^1 l^1
ρ_1: a^1 b^1 c^1 d^1 e^1 f^1 g^1 h^1 l^1 m^1	ρ_1: a^1 b^1 c^1 d^1 e^1 f^1 g^1 h^1 l^1 m^1
(a) A0	(b) A1

Figure 4.3: A sequence of revisions, with origin labeled according to algorithms A0 and A1, with rarity equal to length and threshold 3. In this sequence, content is first deleted and replaced with spam, then almost entirely restored.

origin of $t_0, t_1, \ldots, t_k$ in ρ_n according to the origin of the matching tokens (taking the minimum, in case the longest prefix appears multiple times). The algorithm then proceeds searching for the longest prefix of the remaining unlabeled portion $t_{k+1}, t_{k+2}, \ldots, t_{m-1}$. If no longest prefix can be found, or if the longest prefix from t_0 has rarity below the threshold, then t_0 is labeled as having origin n, or $\Theta(n,0) := n$, and the search continues from the remaining unlabeled portion $t_1, t_2, \ldots, t_{m-1}$. The process continues until the whole of ρ_n has been labeled according to its origin.

Figure 4.4 compares the origin labelings computed by Algorithms A1 and A2. We see

Algorithm A2 Origin via Tichy matching with all preceding revisions.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

1: **for** revisions $n = 1, 2, 3, \ldots$ **do**
2: $k := 0$
3: **while** $k < len(\rho_n)$ **do**
4: Search in $\rho_0, \ldots, \rho_n$ for the longest matching prefixes of $t_k, t_{k+1}, \ldots, t_{len(\rho_n)-1}$. Let $t_k, \ldots, t_m$ be the longest matched prefix, and let $\mathcal{A} = \{(n_1, k_1), \ldots, (n_p, k_p)\}$ be the (possibly empty) set of pairs where the longest matches occur.
5: **if** $\mathcal{A} \neq \emptyset \wedge \gamma(t_k, \ldots, t_m) \geq \Delta$ **then**
6: **for** $i \in \{0, 1, \ldots, m-k\}$ **do**
7: $\Theta(n, k+i) := \min_{1 \leq j \leq p} \Theta(n_j, k_j + i)$
8: **end for**
9: $k := m + 1$
10: **else**
11: $\Theta(n, k) := n$
12: $k := k + 1$
13: **end if**
14: **end while**
15: **end for**

ρ_4:	a^3	b^3	c^1	d^1	a^1	m^1
ρ_3:	a^3	b^3	c^2	d^2	g^2	h^2
ρ_2:	c^2	d^2	g^2	h^2		
ρ_1:	c^1	d^1	a^1	m^1		

(a) A1

ρ_4:	a^3	b^3	c^2	d^2	a^4	m^4
ρ_3:	a^3	b^3	c^2	d^2	g^2	h^2
ρ_2:	c^2	d^2	g^2	h^2		
ρ_1:	c^1	d^1	a^1	m^1		

(b) A2

Figure 4.4: A sequence of revisions, as labeled by Algorithms A1 and A2 with rarity equal to length and threshold 3.

that Algorithm A2 attributes to the tokens `c d a m` in ρ_4 origins `2 2 4 4`, even though these tokens constituted the first revision $\rho 1$. The attribution `1 1 1 1` computed by A1 seems more appropriate.

4.4.4 Properties

Given a sequence of revisions $\rho_0, \rho_1, \rho_2, \ldots$ and two origin labelings Θ, Θ', we write $\Theta \leq \Theta'$ if $\Theta(n,k) \leq \Theta'(n,k)$ at all positions n,k of the sequence; we write $\Theta < \Theta'$ if $\Theta \leq \Theta'$, and if there is at least a position (n,k) where $\Theta(n,k) < \Theta'(n,k)$. The following property establishes that, among A0, A0M, and A1, Algorithm A1 computes the earliest attribution and A0M the latest.

Property 1. *Let Θ_{A0}, Θ_{A0M}, and Θ_{A1} be origin labelings computed by Algorithms A0, A0M, and A1 respectively for a sequence of revisions. Then, $\Theta_{A1} \leq \Theta_{A0} \leq \Theta_{A0M}$. Moreover, there are sequences of revisions for which each of two above inequalities is strict.*

Proof. The weak inequalities follow from the fact that, in deriving the label of a token, the matches considered by A0M are a subset of those considered by A0, which are in turn a subset of those considered by A1. The fact that the inequalities can be strict is witnessed by Figure 4.1

and 4.2. ∎

If a revision occurs twice in the revision history, Algorithm A1 assigns to the later occurrence of the revision an origin labeling that is no greater than that of the first occurrence; the labeling can in fact be strictly smaller. This property makes precise the fact that Algorithm A1, as well as its efficient implementation A3 presented in the next section, are resistant to vandalism. Indeed, consider a sequence $\rho_0, \ldots, rev_i, \ldots, \rho_k$ of revisions, where vandals try to "steal authorship" of content by producing revisions $\rho_{i+1}, \rho_{i+2}, \rho_{k-1}$, until the page is finally restored to its good prior state $\rho_i = \rho_k$. The property implies that no content in ρ_k becomes attributed to any of the vandal authors of $\rho_{i+1}, \ldots, \rho_{k-1}$, nor to the user who restored the previous state of the page.[2] The property holds not only for entire revisions, but also for any sufficiently rare portion of its content.

Property 2. *Consider a sequence of revisions* $\rho_0, \ldots, \rho_i, \ldots, \rho_k$*, and a match* $M = (\rho_i[l : m] = \rho_k[l' : m'])$ *that is sufficiently interesting. Let* Θ *be the origin labeling computed by A1. Then,* $\Theta(k, m + j) \leq \Theta(i, m' + j)$ *for all* $0 \leq j < m - l$*, and there are cases where the inequality can be strict.*

Proof. The result follows from the fact that the matches for $\rho_k[l' : m']$ include $\rho_i[l : m]$. The fact that the inequality can be strict is illustrated in Figure 4.5. ∎

[2] In fact, this statement is true only under the assumption that the entire content of $\rho_i = \rho_k$ has a rarity above the threshold chosen for Algorithm A1. This is the usual case in practice, and can be made the case always by including start and end markers in each revision, and considering interesting any match that includes such end markers.

ρ_5: a^3 b^3 c^1 d^1 g^2 h^2

ρ_4: a^3 b^3 c^1 d^1 a^1 m^1

ρ_3: a^3 b^3 c^2 d^2 g^2 h^2

ρ_2: c^2 d^2 g^2 h^2

ρ_1: c^1 d^1 a^1 m^1

Figure 4.5: A sequence of revisions, as labeled by Algorithm A1 with rarity equal to length and threshold 3. Note that $\rho_5 = \rho_3$, yet the origin labels for some tokens in ρ_5 are smaller than the corresponding ones in ρ_3.

4.5 Efficient Algorithms

In the previous section, we presented various conceptual algorithms for attributing origin to versioned content. The algorithms presented there are extremely inefficient, and have conceptual value only. In this section, we examine the question of efficient implementation for these conceptual algorithms.

Input size and change size. Given a sequence of revisions $\overline{\rho} = \rho_0, \rho_1, \ldots, \rho_n$, the input size for our attribution algoirthms is $|\overline{\rho}| = \sum_{i=0}^{n} len(\rho_i)$ (assuming that tokens can be represented in constant space). In revisioned content, it is often the case that only a small portion of the content is modified at each revision, so that consecutive revisions differ only in a few tokens. It is thus insightful to study the performance of the algorithms not only as a function of the size of the input, but also as a function of the size of the change that occurred. To this end, given two consecutive revisions ρ, ρ', we define $\Delta(\rho, \rho') = \sum_{i=1}^{m} |\beta_i| + \sum_{i=1}^{m} |\gamma_i|$, where $\beta_1, \ldots, \beta_k$

and $\gamma_1, \ldots, \gamma_m$ are the shortest sequences so that we can write:

$$\rho = \alpha_0\beta_1\alpha_1\beta_2\alpha_2\cdots\beta_n\alpha_n$$

$$\rho' = \alpha_0\gamma_1\alpha_1\gamma_2\alpha_2\cdots\gamma_n\alpha_n$$

In other words, we write ρ and ρ' as composed of maximal sequences of unchanged portions of text $\alpha_0, \ldots, \alpha_m$, and of portions $\beta_1, \ldots, \beta_m$ in ρ that will be replaced by sequences $\gamma_1, \ldots, \gamma_m$ in ρ'. We then define the *change size change*$(\overline{\rho})$ of $\rho_0, \rho_1, \ldots, \rho_n$ as *change*$(\overline{\rho}) = \sum_{i=0}^{n-1} \Delta(\rho_i, \rho_{i+1})$.

Summary size and one-revision update. Revisioned content is produced, as the name implies, one revision at a time. When computing the origin of the tokens in the newest revision ρ_n, it would be impractical to read and re-process all previous revisions $\rho_0, \ldots, \rho_{n-1}$. Practical algorithms rely on a *summary* $\mathcal{S}_{n-1}$ of $\rho_0, \ldots, \rho_{n-1}$, containing all the information that the algorithm needs to know about the preceding revisions to attribute later revisions. The algorithms compute the origin labeling for ρ_n on the basis of $\mathcal{S}_{n-1}$ and ρ_n, producing as output both $\mathcal{S}_n$ and the origin labeling for ρ_n. We refer to this computation as the *one-revision update.* We will thus study how the summary size, and the running time for the one-revision update depend on the input size and change size.

4.5.1 Algorithm A3

Consider a fixed a rarity function γ and a rarity threshold Δ. We say that a sequence of tokens $t_1, t_2, \ldots, t_n$ is *minimally interesting* if $\gamma(t_1, t_2, \ldots, t_n) \geq \Delta$, and at least one of $\gamma(t_2, \ldots, t_n) < \Delta$ or $\gamma(t_1, \ldots, t_{n-1}) < \Delta$ holds. When the rarity function is simply the number of tokens, and the

rarity threshold Δ is an integer, then the minimally interesting sequences are the sequences consisting of Δ tokens. We say that a match $M = (\rho_n[i:j] = \rho_{n'}[i':j'])$ is *minimally interesting* if $\rho_n[i], \rho_n[i+1], \ldots, \rho_n[j-1]$ is minimally interesting. To obtain an efficient implementation of Algorithm A1, we start from the observation that in Step 3 of Algorithm A1, we need to consider only minimally interesting matches.

Lemma 1. *If in Step 3 of Algorithm A1 the set $\widehat{\mathcal{M}}$ is limited only to minimally interesting matches, the labeling computed by the algorithm is unchanged.*

Proof. For a token t_k of ρ_n, let M be a match realizing the minimum in Step 7 of Algorithm A1, and let $t_i, \ldots, t_j$ be the matched sequence, with $i \leq k \leq j$. If M is minimally interesting, the result holds. If M is not minimally interesting, then both sub-matches for $t_i, \ldots, t_{j-1}$ and $t_{i+1}, \ldots, t_j$ are interesting, and t_k belongs to one of them. Continuing in this fashion, we can find a submatch M' of M that contains t_k and that is minimally interesting. Since t_k would be assigned the same origin under M or M', the result holds. ∎

This result suggests implementing Algorithm A1 in terms of a *trie*. A trie $\mathcal{T}$ is a tree whose edges are labeled with tokens, and such that the edges outgoing from a node are labeled by distinct tokens. We say that a sequence of tokens $t_1, t_2, \ldots, t_m$ *belongs* to the trie $\mathcal{T}$, written $t_1, t_2, \ldots, t_m \in \mathcal{T}$, if there is a path from the root labeled with the sequence, and we use the sequence to refer to the node where the path ends. If the sequence $t_1, t_2, \ldots, t_m$ is minimally interesting, we say that the corresponding node is minimally interesting. If $\gamma = \mathit{len}$ and Δ is an integer, the minimally interesting nodes are those at depth Δ in the trie. We denote by $\perp$ the empty trie consisting only of a root node, and we denote by $\mathit{Ins}(\mathcal{T}; t_1, \ldots, t_m)$ the

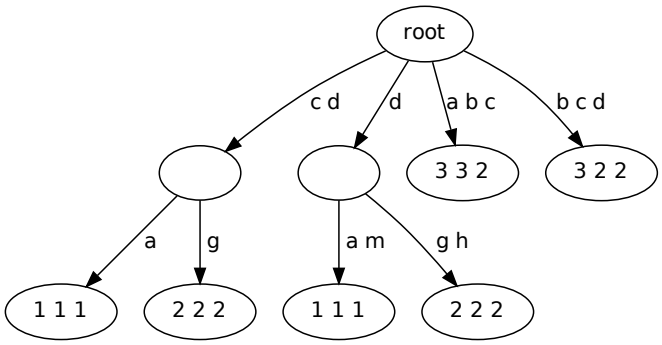


Figure 4.6: Trie resulting after processing revisions ρ_1, ρ_2, ρ_3 as in Figure 4.4.

result of creating a path labeled by $t_1, \ldots, t_m$ in $\mathcal{T}$ in the trie. In the implementation of A1, we use tries to represent all the minimally interesting sequences of tokens that have occurred in past revisions. Each minimally interesting node $t_1, \ldots, t_m$ of the trie is labeled with the origin $k_1, \ldots, k_m = \ell(t_1, \ldots, t_m)$ of the sequence of tokens $t_1, \ldots, t_m$. This yields Algorithm A3.

Figure 4.6 illustrates the trie resulting after processing revisions ρ_1, ρ_2, ρ_3 as in Figure 4.4, for a rarity function equal to length, and threshold 3. The leaf nodes are the minimally interesting nodes. To save space in the trie, we omit the non-interesting nodes that have a single child, concatenating the labels of the edges leading into and out of such nodes.

The following theorem shows that Algorithms A3 and A1 compute the same origin labels.

Theorem 1. *Algorithm A3 computes the same origin labels as Algorithm A1.*

To state the proof of this theorem, consider a sequence $\sigma = t_1, \ldots, t_k$ occurring at least once in a set of revisions $\rho_0, \ldots, \rho_m$ that has been labeled according to its origin by Algorithm A1. For $1 \leq j \leq k$, let p_j be the minimum label that token t_j is assigned in any of these

Algorithm A3 Implementation of A1 in terms of tries.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

1: $\mathcal{T} := \perp$
2: **for** revisions $n = 1, 2, 3, \ldots$ **do**
3: **for all** positions $0 \leq k < len(\rho_n)$ of ρ_n **do**
4: $\Theta(n,k) := n$
5: **end for**
6: **for all** minimally interesting sequences $t_k, \ldots, t_m$ of ρ_n **do**
7: **if** $t_k, \ldots, t_m \in \mathcal{T}$ **then**
8: $i_k, \ldots, i_m := \ell(t_k, \ldots, t_m)$
9: **for all** $j \in [k, \ldots, m]$ **do**
10: $\Theta(n,j) := \min\{\Theta(n,j), i_j\}$
11: **end for**
12: **end if**
13: **end for**
14: **for all** minimally interesting sequences $t_k, \ldots, t_m$ of ρ_n **do**
15: **if** $t_k, \ldots, t_m \in \mathcal{T}$ **then**
16: $\ell(t_k, \ldots, t_m) := \Theta(n,k), \ldots, \Theta(n,m)$
17: **else**
18: $\mathcal{T} := Ins(\mathcal{T}; t_k, \ldots, t_m)$
19: $\ell(t_k, \ldots, t_m) := \Theta(n,k), \ldots, \Theta(n,m)$
20: **end if**
21: **end for**
22: **end for**

occurrences. We say that $p_1, \ldots, p_k$ is the *minimal labeling* of σ in $\rho_0, \ldots, \rho_m$.

Proof. The proof proceeds by induction, using the inductive hypothesys that, after processing revisons $\rho_0, \ldots, \rho_n$, the trie $\mathcal{T}$ contains exactly all the minimally interesting sequences occurring in $\rho_0, \ldots, \rho_n$, each labeled with its minimal labeling in $\rho_0, \ldots, \rho_n$.

Assume that Algorithm A3 has processed $\rho_0, \ldots, \rho_{n-1}$ already, and is processing ρ_n.

First, we show that this inductive hypothesis implies that algorithms A1 and A3 produce the same labeling. There are two directions to the argument.

- Assume that Algorithm A1 assigns origin label p to token $\rho_n[k]$. Let $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$ be the minimally interesting match for which the minimum in Line 7 is realized (this exists, due to Lemma 1). By induction hypothesis, the trie $\mathcal{T}$ will contain the sequence $\rho_{n'}[i' : j']$ with its minimal labeling, in which the token $\rho_n[k]$ is labeled with origin p. Thus, Algorithm A3 in Steps 3–13 will assign to $\rho_n[k]$ an origin no larger than p.

- Conversely, assume that Algorithm A3 assigns origin p to token $\rho_n[k]$. Then, $\mathcal{T}$ must have contained a minimally interesting sequence $\rho_n[j : l] = t_j, \ldots, t_{l-1}$, for $j \leq k < l$, where t_k is labeled by p. By inducton hypothesis, p is the minimal label of t_k in all occurrences of $t_j, \ldots, t_{l-1}$ in $\rho_0, \ldots, \rho_{n-1}$, indicating that Algorithm A1 also labels $\rho_n[k]$ with label no greater than p.

Second, we show that once ρ_n is processed by A3, the induction hypothesis holds also for $\rho_0, \ldots, \rho_n$. Consider a minimally interesting sequence σ occurring in ρ_n (the situation of minimally interesting sequences not occurring in ρ_n is unchanged). The arguments in the first

part of this proof ensure that once Steps 3–13 have terminated, the sequence σ in ρ_n is labeled according to its minimal labeling. Steps 14–21 ensure then that the sequence σ is present in the trie $\mathcal{T}$, and is labeled in it according to its minimal labeling. This completes the induction step.

∎

The following theorem characterizes the time and space requirements for Algorithm A3.

Theorem 2. *If there is an integer M such that all token sequences of length at least M are interesting, then given a sequence $\rho_0, \rho_1, \rho_2, \ldots$ of revisions, Algorithm A3 can perform a one-revision update for revision ρ_n using a summary of size $O(change(\rho_0, \ldots, \rho_{n-1}))$, and in time $O(len(\rho_n))$.*

Proof. Algorithm A3 uses as summary for $\rho_0, \ldots, \rho_{n-1}$ the trie $\mathcal{T}_{n-1}$ resulting from the processing of these revisions. To prove the space requirement, we can prove by induction over n that $|\mathcal{T}_n| \leq K \cdot change(\rho_0, \ldots, \rho_n)$, for some fixed $K \geq 0$. Note that M is a bound for the length of any minimally interesting sequence: in fact, any interesting sequence σ longer than M has its leftmost M tokens, and rightmost M tokens, also form interesting sequences, contradicting the minimality of σ. Let $K = M(M+1)/2$ be the maximum number of sequences of length at most M that contain a given position. Note that a single insertion or deletion going from ρ_{n-1} to ρ_n affects at most K minimally interesting sequences in ρ_n. Therefore, at most $K \cdot \Delta(\rho_{n-1}, \rho_n)$ new minimally interesting sequences are going to be inserted in $\mathcal{T}_{n-1}$ in order to obtain $\mathcal{T}_n$. This leads to the space bound for the summary.

To prove the time bound for the processing of ρ_n, it suffices to note that there are at most $len(\rho_n)$ minimally interesting matches involving ρ_n, and that processing each one of them

(including accessing the trie for retrieving the minimal labeling of any match) takes constant time (the trie has depth at most K). ∎

Note that the theorem implies that the processing of a sequence $\rho_0, \ldots, \rho_n$ of revisions can be done in time $O(|(|\rho_0, \ldots, \rho_n))$.

If the rarity of a sequence of tokens is taken to be its length, then trivially all sequences longer than the rarity threshold are rare. Another case when the length of minimally interesting sequences of tokens is bounded is when the rarity of a sequence of tokens $t_1, \ldots, t_k$ is computed as $\gamma(t_1, \ldots, t_k) = \prod_{i=1}^{k} \frac{1}{p_{k_i}}$ for some token probabilities $0 \leq p_{k_i} \leq 1$, and if there is an upper bound $c < 1$ for the probability of any token.

These results suggest that Algorithm A3 is optimal: it is not possible to label a sequence of revisions in time less than the input size, and it is not possible to label a new revision storing less information about the past than all change that has occurred (except if compression techniques are used; such techniques can also be applied to the representation of our trie summaries).

In large-scale implementations of origin analysis, the summary of a revision sequence cannot be stored permanently in-memory: rather, it must be read from persistent storage (such as a database) before the algorithm analyzes a new revision, and written back to persistent storage once the analysis is done. If the time to read and write the summary is included, then the time required for analyzing revision ρ_n of sequence $\rho_0, \rho_1, \rho_2, \ldots$ is in $O(len(\rho_n) + change(\rho_0, \ldots, \rho_n))$.

4.5.2 Tichy matching

The Tichy-based Algorithm A2 is defined in terms of longest common matches. We can obtain an efficient implementation in terms of *suffix trees,* which provide the most time efficient implementation of the longest common substring problem [40]. A suffix tree is a tree-like data structure that can represent all the suffixes $a_i, a_{i+1}, a_{i+2}, \ldots, a_m$, $0 \leq i < m$, of a given string $a_0, a_1, \ldots, a_m$; they can be constructed in time linear in the length of the string [81, 55, 77]. The construction of suffix trees can be adapted so that $\mathcal{S}_{n-1}$ is a suffix tree representing all the suffixes of $\rho_0, \rho_1, \ldots, \rho_{n-1}$; see [40] for similar adaptations. This leads to Algorithm A2s. The origin information can be associated with the suffix tree in similar fashion to what was done for the trie; we omit the details to conserve space. The drawback of this algorithm, compared to A3, is that the size required by the summary is proportional to the size of all previous revisions, rather than to the change size. This because a change involving a token in the middle of a revision of length m gives rise to $m/2$ new suffixes on average, each of which corresponds to at least one new suffix tree node. Figure 4.7 illustrates this: the change from "deer" to "dear" gives rise to three new suffixes, corresponding to nodes 8, 9, and 10.

Theorem 3. *Let $M = |\rho_0, \ldots, \rho_n|$ and $D = change(\rho_0, \ldots, \rho_n)$. Algorithm A2s produces the origin labels for revision ρ_n in time $O(M)$; the time for labeling the complete sequence $\rho_0, \ldots, \rho_n$ is $O(M^2)$. There are some examples of input for which the running time for ρ_n exceeds $K \cdot D$, for any $K \geq 0$, so that the running time is not $O(D)$. The size of $\mathcal{S}_n$ is $O(M)$, and is not in $O(D)$.*

Proof. The space and time results are a consequence of the results on the construction of suffix trees [81, 55, 77, 40]. The existence of sequences in which the summary size is proportional

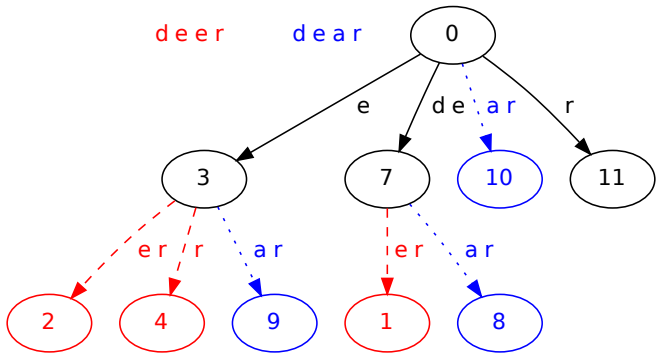


Figure 4.7: Suffix tree for two string "deer" and "dear". Solid edges correspond to both strings; dashed edges correspond to "deer"; dotted edges correspond to "dear". We omit for clarity the unique terminal symbols that are added to each string.

to the entire input size, rather than to the change size, follows from the fact that changing a single token in a revision of length m leads to the creation of a number of new suffixes that is proportional to m (on average, equal to $m/2$). These new suffixes must be represented in the suffix tree, so that $\mathcal{S}_n$ is in $O(|\rho_0,\ldots,\rho_n|)$ but not in $O(change(\rho_0,\ldots,\rho_n))$. ∎

4.6 Experimental Results

We have produced a robust, scalable implementation of Algorithm A3 that can be applied to very large wikis, including the English Wikipedia. Each revision is parsed in a sequence of tokens, which consists of white-space separated sequences of non-whitespace characters: tokens thus loosely correspond to words. This tokenization step could be improved by considering the structure of the MediaWiki markup language. We do not use individual (unicode) charac-

Algorithm A2s Origin via Tichy matching with all preceding revisions, implemented via suffix trees.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.

Output: An origin labeling Θ for $\overline{\rho}$.

Let $\mathcal{S}_0$ be the empty suffix tree.
for revisions $n = 1, 2, 3, \ldots$ **do**
  $k := 0$
  while $k < len(\rho_n)$ **do**
    Search in $\mathcal{S}_{n-1}$ for the longest matching prefixes of $t_k, t_{k+1}, \ldots, t_{len(\rho_n)-1}$. Let $t_k, \ldots, t_m$ be the longest matched prefix, and let $i_1, \ldots, i_k$
    if $\mathcal{A} = \emptyset \vee \gamma(t_k, \ldots, t_m) \geq \Delta$ **then**
      for $i \in \{0, 1, \ldots, m-k\}$ **do**
        $\Theta(n, k+i) := \min_{1 \leq j \leq p} \Theta(n_j, k_j + i)$
      end for
      $k := m + 1$
    else
      $\Theta(n, k) := n$
      $k := k + 1$
    end if
  end while
  Update $\mathcal{S}_{n-1}$ by adding all the suffixes of of ρ_n, yielding $\mathcal{S}_n$.
end for

ters as our unit of tokenization, for two reasons. First, using words as attribution units tends to produce more natural results, since contributors typically create or rearrange content in word units; word-level attribution is also easier to display via coloring or other visual cues. Second, using individual characters as tokens would lead to a larger size for the trie summary, as the trie would grow deeper.

The algorithm uses as rarity function the length of a token sequence, and a configurable threshold. The algorithm does not use a rarity function that depends on token (word) frequency, chiefly to save space by avoiding the need to store the frequency of a large number of words; we may revisit this decision at a later time. For each wiki page $\mathcal{P}$, the algorithm stores in persistent storage the pair $(n, \mathcal{T}_n)$, consisting of the index n of the last revision of $\mathcal{P}$ that has been processed, along with the labeled trie $\mathcal{T}_n$ representing the summary. When a new revision ρ_m for $\mathcal{P}$ is produced, with $m > n$, the algorithm processes all revisions $\rho_{n+1}, \ldots, \rho_m$: there can be multiple revision to analyze, since the algorithm may have been inactive at times (due to system maintenance), or indeed, it may not have run yet on the page. Each of $\rho_{n+1}, \ldots, \rho_m$ is fed to the algorithm; the algorithm computes and stores the origin of these revisions, and finally stores $(m, \mathcal{T}_m)$ associated with page $\mathcal{P}$.

The code, and a demo of this implementation is available at `https://sites.google.com/a/ucsc.edu/luca/the-wikipedia-authorship-project`, along with all the data used for the experiments reported here. We provide experimental data computed on two revison datasets:

- Dataset A: articles with more than 200 revisions in files wiki-00000066.xml.gz and wiki-00000193.xml.gz. The dataset consists of 78k revisions in 75 articles.

- Dataset B: articles with at least 1000 revisions occurring in files wiki-00000066.xml.gz, wiki-00000193.xml.gz, wiki-00000134.xml.gz and wiki-00000384.xml.gz. The dataset consists in 50 revisions.

The above *.xml.gz files were chosen at random among the first 1000 files obtained by splitting in 100-page portions a 2010 dump of the English Wikipedia. Unless otherwise noted, we provide results for a rarity function equal to length, and threshold 4.

Content aging. In the editing of Wikipedia revisions, it occasionally happens that vandals introduce vast amounts of spurious content. This content is almost immediately removed by editors or non-vandal users. Yet, since our algorithms store a representation of the entire history of each page, that spurious content would persist indefinitely in our trie summary. This would offer an avenue to vandals for severely impacting our performance. To limit this effects of vandalism, our implementation discards content that has not appeared in any recent revision: this is acceptable in practice, since content that has been long removed from a page is unlikely to be re-inserted. To this end, we label every node of the trie $\mathcal{T}$ used in Algorithm A3 with the *node age,* consisting of a pair (N,T). The integer N and the timestamp T represents, respectively, the most recent revision index and the most recent time when the node was traversed by the algorithm. Once Algorithm A3 has processed a revision n produced at time T_n, and before writing back the trie to persistent storage, we prune from the trie all the nodes that have both $n-N > \Delta_N$ and $T_n - T > \Delta_T$, where the thresholds Δ_N and Δ_T are configurable. Table 4.8 compares the difference in attribution and size arising from different aging thresholds. The table was obtained from dataset A.

	attribution difference	trie size
$\Delta_N = 200$, $\Delta_T = 180$ days	1.3 %	70 %
$\Delta_N = 100$, $\Delta_T = 90$ days	2.1 %	55 %

Figure 4.8: Attribution difference, and trie size, for various aging thresholds, as compared to no content aging.

Attribution comparison among A0, A1, A2. Figure 4.9 plots the difference in the attributions computed by Algorithms A0, A1, and A2, for a rarity function equal to token sequence length, and various values of rarity threshold. These comparisons have been done without using any age-driven pruning of trie nodes in Algorithm A1, to make the comparison fair across algorithms. The figure gives the tokens with different attribution, as percentage of the total tokens, for dataset A. As we can see, Algorithm A1 computes an attribution that is over 75% different from the one computed by Algorithm A0. This is due to the fact that Algorithm A0 considers new any content that is re-inserted after a deletion. As an example, Figure 4.10 plots the size of the revisions, and summary trie, for the Wikipedia article on "Dance Dance revolution"; the frequent dips in revision size correspond to content deletions by vandals. From Figure 4.9 we see also that the attribution difference between algorithms A1 and A2 is of only a few percentage points, when the length of minimally interesting matches is 3 or more. The advantage of Algorithm A1 over A2 lies in its efficient implementation.

Size of trie and suffix tree summaries. Figure 4.11 plots the ratio between the size of the trie serialized in Json, and the average size of the last 10 revisions, for aging values $\Delta_N = 100$ and $\Delta_T = 90$ days and dataset B. We use the average size of the last 10 revisions, rather than the size of the last revision, to avoid very large spikes in the ratio when the content of a revision is

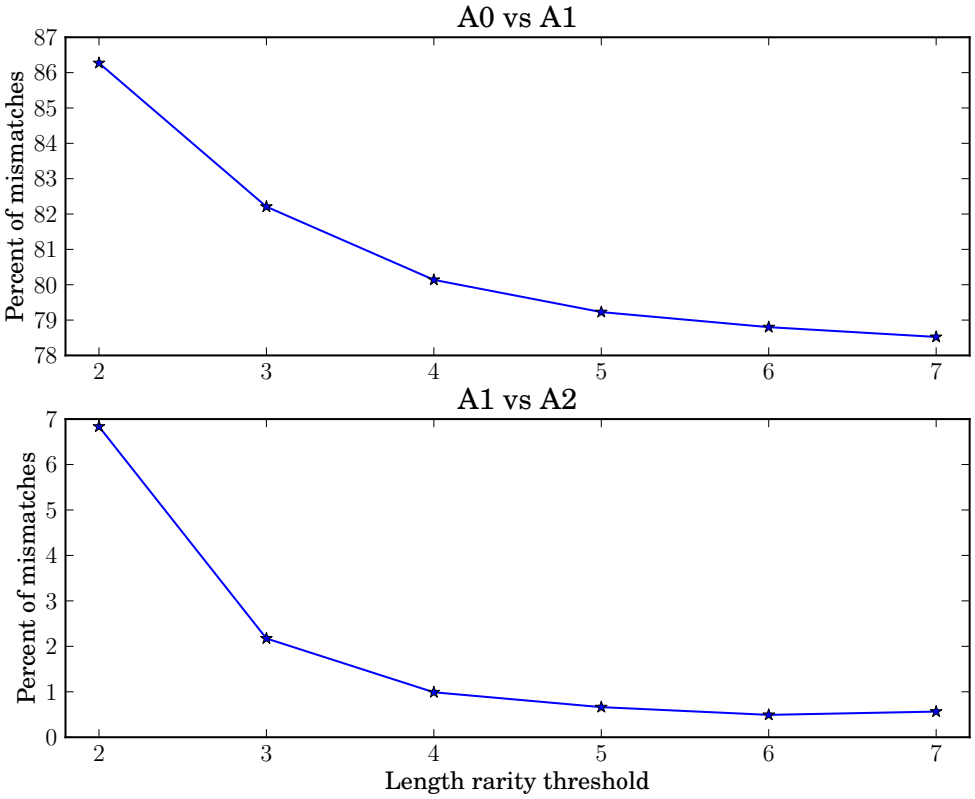


Figure 4.9: Difference between attribution by A0, A1 and A2 for length rarity function with various threshold.

deleted by vandals. The average ratio is approximately 10; the ratio can be reduced to about 3 by compressing the trie serializations with gzip. This is a very practical amount of storage, which is dwarfed in the English Wikipedia by the amount of storage required to store all revisions of every page.

In Figure 4.12 we compare the size of the trie summaries used by Algorithm A3, with the size of the suffix tree summaries used in implementing Algorithm A2. Dataset B was used, and no content aging was applied, to make the comparison fair. The trie sizes are tied to the change between revisions, and since we discard text that has been dead for long, they tend to be

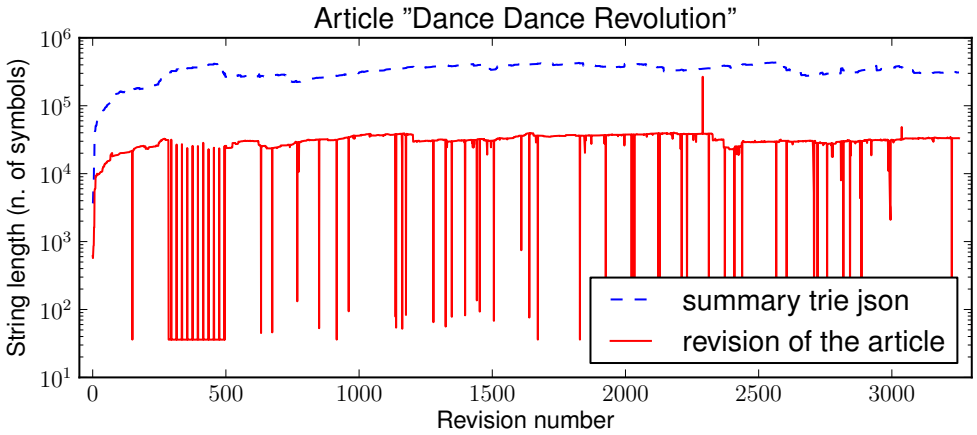


Figure 4.10: Length of revisions (in number of characters) of the article "Dance Dance Revolution" compare to length of json string with the trie summary. Dips in the revision size indicate content deletions due to vandalism.

a constant multiple of the revision size. On the other hand, the suffix tree sizes are proportional to the total size of past revisions.

Time performance. Figure 4.13 summarizes the time performance of Algorithm A3, as a function of revision size. In the figure, the *algorithm time* is the time required by steps 3–21, as well as content aging, of A3; the *serialization time* is the time required for serializing and deserializing the trie into json. As we see, these two times are of the same order of magnitude, indicating that there is limited scope for improvement by optimizing the implementation of steps 3–21. The figure was obtained using dataset A.

4.7 Conclusions

We have considered so far revisioned content that consists in a single revisioned entity. Most revisioned content, however, consists of multiple entities: a national Wikipedia con-

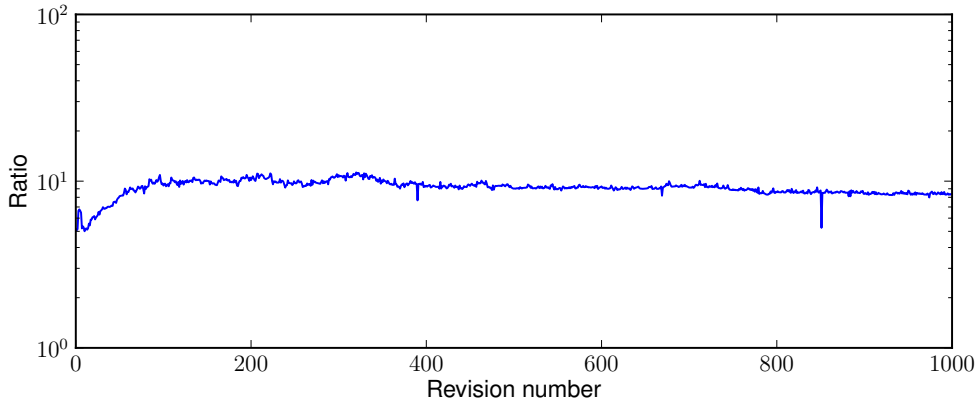


Figure 4.11: Ratio between the trie summary size and the average size of the last 10 revisions.

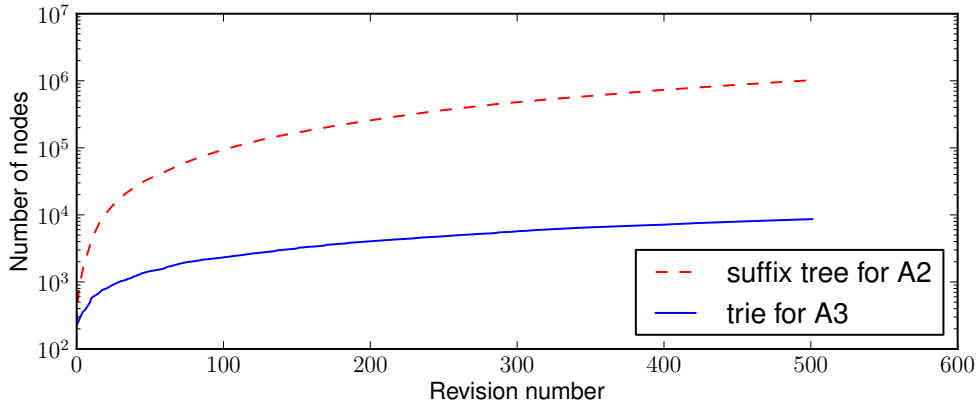


Figure 4.12: Size comparison between trie summaries for A3 and suffix tree summaries for A2.

sists of a set of pages, each of which is versioned, and a code repository similarly consists of multiple files, each revisioned. Furthermore, in modern revisioning systems such as git (`http://git-scm.com`), the various revisions are organized in branches. Since code is commonly copied across files, and to a lesser extent, content is moved across Wikipedia pages, an origin analysis that spans a whole repository is often desirable.

We can perform such repository-wide analysis with the algorithms we discussed in this paper, by considering the stream of *all* revisions $\rho_0, \rho_1, \rho_2, \ldots$ in the order they are created, regardless of the entity (page, or file, or branch) to which they belong. The content of each

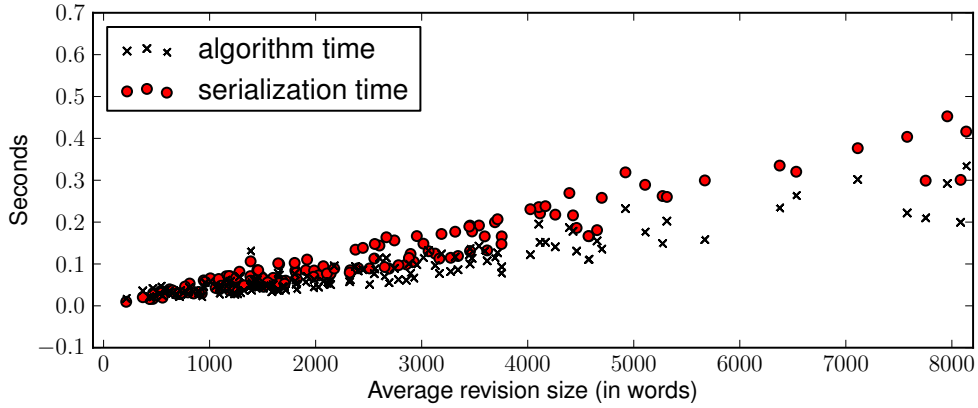


Figure 4.13: Time performance of algorithm A3. Each point in the plot represents an article, with the average revision size on the X axis. The times required by attribution computation, and trie serialization and deserialization, are reported on the Y axis.

new revision will be compared with all previous content, assigning origin via matching with corresponding occurrences. The algorithms could be improved by considering as more likely the matches that relate different versions of the same entity, as compared to matches that relate different entities. For software repositories, which are of moderate size, and where revisions are typically generated at low speed (even large industrial code bases have intervals between revisions of several seconds), such a global origin analysis would be feasible. In the English Wikipedia, however, several revisions per second may be created. From our experimental data, the size of a global summary would about ten times the cumulative size of the most recent revisions of all pages, leading to a size of approximately one terabyte. This size exceeds the RAM memory easily available in a single, low-cost host. The design of a system capable of comparing, in real time, every revision of Wikipedia with the whole of its past history would be challenging, and the result expensive to operate. For this reason, in our implementation we have opted to compare new revisions only with the previous content of the page to which the

revisions belong. If required, we will address content moved across pages via specialized tools.

Compared to the algorithm of [29], the one presented here offers a simple mathematical definition of authorship, is applicable to any revisioned content, comes with compexity bounds and robustness characterization, and is well-suited to an implementation in which the authorship information needs to be computed on-line, as revisions are made. Unfortunately, we became aware of the work of [29] too late to be able to present here a quantitative comparison of how well the two algorithms match a human perception of authorship on the Wikipedia.

Bibliography

[1] B. T. Adler. Wikitrust: content-driven reputation for the wikipedia. 2012.

[2] B. T. Adler and L. de Alfaro. A content-driven reputation system for the wikipedia. In *Proceedings of the 16th international conference on World Wide Web*, pages 261–270. ACM, 2007.

[3] B. T. Adler, L. de Alfaro, I. Pye, and V. Raman. Measuring author contributions to the wikipedia. In *Proceedings of the 4th International Symposium on Wikis*, page 15. ACM, 2008.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 684–693, New York, NY, USA, 2005. ACM.

[5] N. Alon, F. Fischer, A. Procaccia, and M. Tennenholtz. Sum of us: Strategyproof selection from the selectors. In *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 101–110. ACM, 2011.

[6] K. Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58:328, 1950.

[7] J. Bartholdi, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[8] S. Bhatnagar, M. Desmarais, C. Whittaker, N. Lasry, M. Dugdale, and E. S. Charles. An analysis of peer-submitted and peer-reviewed answer rationales, in an asynchronous peer instruction based learning environment. *Proceedings of the 8th International Conference on Educational Data Mining*, 2015.

[9] R. Bradley and M. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):pp. 324–345, 1952.

[10] P. Buneman, S. Khanna, and W.-C. Tan. Data provenance: Some basic issues. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 87–93. Springer, 2000.

[11] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In *International conference on database theory*, pages 316–330. Springer, 2001.

[12] Y. Cai, C. Daskalakis, and C. H. Papadimitriou. Optimum statistical estimation with strategic data sources. *ArXiv e-prints*, 2014.

[13] G. A. Calvo and S. Wellisz. Supervision, loss of control, and the optimum size of the firm. *Journal of political Economy*, 86(5):943–952, 1978.

[14] G. A. Calvo and S. Wellisz. Hierarchy, ability, and income distribution. *Journal of political Economy*, 87(5, Part 1):991–1010, 1979.

[15] A. Carvalho, S. Dimitrov, and K. Larson. Inducing honest reporting without observing outcomes: An application to the peer-review process. *arXiv preprint arXiv:1309.3197*, 2013.

[16] M. Cisel, R. Bachelet, and E. Bruillard. Peer assessment in the first french mooc: Analyzing assessors' behavior. *Proceedings of the 7th International Conference on Educational Data Mining*, 2014.

[17] R. T. Clemen. Incentive contracts and strictly proper scoring rules. *Test*, 11(1):167–189, 2002.

[18] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977, 2001.

[19] A. Das Sarma, A. Das Sarma, S. Gollapudi, and R. Panigrahy. Ranking mechanisms in twitter-like forums. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 21–30. ACM, 2010.

[20] A. Dasgupta and A. Ghosh. Crowdsourced judgement elicitation with endogenous proficiency. In *Proceedings of the 22nd international conference on World Wide Web*, pages 319–330. International World Wide Web Conferences Steering Committee, 2013.

[21] A. Dawid and A. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, pages 20–28, 1979.

[22] L. de Alfaro, A. Kulshreshtha, I. Pye, and B. Adler. Reputation systems for open collaboration. *Communications of the ACM*, 54(8):81–87, 2011.

[23] J.-C. de Borda. *Memoire sur les Elections au Scrutin*. 1781.

[24] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[25] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 613–622. ACM, 2001.

[26] A. Elo. *The Rating of Chess Players Past and Present*. New York, Arco, 1978.

[27] A. G. Erdman and G. N. Sandor. *Mechanism design: analysis and synthesis (Vol. 1)*. Prentice-Hall, Inc., 1997.

[28] B. Faltings, J. J. Li, and R. Jurca. Eliciting truthful measurements from a community of sensors. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 47–54. IEEE, 2012.

[29] F. Flöck and A. Rodchenko. Whose article is it anyway?–detecting authorship distribution in wikipedia articles over time with wikigini. *Online proceedings of the Wikipedia Academy*, 2012.

[30] A. Forte and A. Bruckman. Why do people write for wikipedia? incentives to contribute to open–content publishing. *Proc. of GROUP*, 5:6–9, 2005.

[31] N. S. Foundation. Dear colleague letter: Information to principal investigators (pis) planning to submit proposals to the sensors and sensing systems (sss) program October 1 , 2013 deadline, 2013.

[32] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3), 2008.

[33] A. Gao, J. R. Wright, and K. Leyton-Brown. Incentivizing evaluation via limited access to ground truth: Peer-prediction makes things worse. *arXiv preprint arXiv:1606.07042*, 2016.

[34] J. Geanakoplos. Three brief proofs of arrows impossibility theorem. *Economic Theory*, 26(1):211–215, 2005.

[35] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

[36] A. Ghosh. Game theory and incentives in human computation systems. In *Handbook of Human Computation*, pages 725–742. Springer, 2013.

[37] M. E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, 1999.

[38] M. Glickman. *Paired Comparison Models with Time-varying Parameters*. Harvard University, 1993.

[39] C. Godsil and G. F. Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

[40] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[41] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.

[42] C. Harris. Youre hired! an examination of crowdsourcing incentive models in human resource tasks. In *Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 15–18, 2011.

[43] R. Jin and Z. Ghahramani. Learning with multiple labels. In *Advances in neural information processing systems*, pages 897–904, 2002.

[44] S. Johnson, J. W. Pratt, and R. J. Zeckhauser. Efficiency despite mutually payoff-relevant private information: The finite case. *Econometrica: Journal of the Econometric Society*, pages 873–900, 1990.

[45] R. Jurca and B. Faltings. Enforcing truthful strategies in incentive compatible reputation mechanisms. In *Internet and Network Economics*, pages 268–277. Springer, 2005.

[46] R. Jurca, B. Faltings, et al. Mechanisms for making crowds truthful. *Journal of Artificial Intelligence Research*, 34(1):209, 2009.

[47] V. Kamble, N. Shah, D. Marn, A. Parekh, and K. Ramachandran. Truth serums for massively crowdsourced evaluation tasks. *arXiv preprint arXiv:1507.07045*, 2015.

[48] D. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *Proc. of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, 2011.

[49] M. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Edward Arnold, 1990.

[50] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 95–103, New York, NY, USA, 2007. ACM.

[51] Y. Kong, G. Schoenebeck, and K. Ligett. Putting peer prediction under the micro (economic) scope and making truth-telling focal. *arXiv preprint arXiv:1603.07319*, 2016.

[52] C. Kulkarni, K. P. Wei, H. Le, D. Chia, K. Papadopoulos, J. Cheng, D. Koller, and S. R. Klemmer. Peer and self assessment in massive online classes. In *Design Thinking Research*, pages 131–168. Springer, 2015.

[53] A. Lijphart. *Electoral Systems and Party Systems: A Study of Twenty-Seven Democracies, 1945, 1990*. Oxford University Press, 1994.

[54] R. Luce. *Individual choice behavior : a theoretical analysis*. Wiley N.Y, 1959.

[55] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[56] M. Merrifield and D. Saari. Telescope time without tears: A distributed approach to peer review. *Astronomy & Geophysics*, 50(4):4–16, 2009.

[57] P. Michelucci. *Handbook of human computation*. Springer, 2013.

[58] N. Miller, P. Resnick, and R. Zeckhauser. Eliciting informative feedback: The peer-prediction method. *Management Science*, 51(9):1359–1373, 2005.

[59] L. Moreau, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, et al. The provenance of electronic data. *Communications of the ACM*, 51(4):52–58, 2008.

[60] P. Naghizadeh and M. Liu. Incentives, quality, and risk: A look into the nsf proposal review pilot. *Arxiv*, 1307.6528v1, 2013.

[61] O. Nov. What motivates wikipedians? *Communications of the ACM*, 50(11):60–64, 2007.

[62] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. *Human computation*, 11(11), 2011.

[63] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[64] C. Piech, J. Huang, Z. Chen, C. Do, A. Ng, and D. Koller. Tuned models of peer assessment in moocs. *arXiv preprint arXiv:1307.2579*, 2013.

[65] C. Piech, J. Huang, Z. Chen, C. Do, A. Ng, and D. Koller. Tuned models of peer assessment in MOOCs. *arXiv preprint arXiv:1307.2579*, 2013.

[66] D. Prelec. A bayesian truth serum for subjective data. *science*, 306(5695):462–466, 2004.

[67] Y. Qian. Incentives and loss of control in an optimal hierarchy. *The Review of Economic Studies*, 61(3):527–544, 1994.

[68] G. Radanovic and B. Faltings. A robust bayesian truth serum for non-binary signals. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, number EPFL-CONF-197486, pages 833–839, 2013.

[69] G. Radanovic and B. Faltings. Incentives for truthful information elicitation of continuous signals. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[70] V. Raykar, S. Yu, L. Zhao, G. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *J. Mach. Learn. Res.*, 11:1297–1322, August 2010.

[71] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[72] B. Riley. Minimum truth serums with optional predictions. In *Proceedings of the 4th Workshop on Social Computing and User Generated Content (SC14)*, 2014.

[73] R. T. Rockafellar. *Convex analysis*. Princeton university press, 2015.

[74] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[75] P. Smyth, U. Fayyad, M. Burl, P. Perona, and P. Baldi. Inferring ground truth from subjective labelling of venus images. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *NIPS*, pages 1085–1092. MIT Press, 1994.

[76] W. F. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems (TOCS)*, 2(4):309–321, 1984.

[77] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[78] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *Proceedings of the first international workshop on crowdsourcing and data mining*, pages 15–21. ACM, 2012.

[79] B. Waggoner and Y. Chen. Output agreement mechanisms and common knowledge. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[80] T. Walsh. The peerrank method for peer assessment. *arXiv preprint arXiv:1405.7192*, 2014.

[81] P. Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

[82] P. Welinder, S. Branson, S. Belongie, and P. Perona. The multidimensional wisdom of crowds. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2424–2432. 2010.

[83] J. Whitehill, T.-F. Wu, J. Bergsma, J. Movellan, and P. Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[84] O. E. Williamson. Hierarchical control and optimum firm size. *Journal of political economy*, 75(2):123–138, 1967.

[85] R. L. Winkler and A. H. Murphy. "Good" probability assessors. *Journal of applied Meteorology*, 7(5):751–758, 1968.

[86] J. Witkowski, Y. Bachrach, P. Key, and D. C. Parkes. Dwelling on the negative: Incentivizing effort in peer prediction. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[87] J. Witkowski and D. C. Parkes. A robust bayesian truth serum for small populations. In *AAAI*, 2012.

[88] W. Xiong, D. J. Litman, and C. D. Schunn. Assessing reviewer's performance based on mining problem localization in peer-review data. In *EDM*, pages 211–220. ERIC, 2010.

[89] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Exploring artificial intelligence in the new millennium. chapter Understanding belief propagation and its generalizations, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[90] P. Zhang and Y. Chen. Elicitability and knowledge-free elicitation with peer prediction. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent*

systems, pages 245–252. International Foundation for Autonomous Agents and Multiagent Systems, 2014.