Helium 2.0 User Manual

Harvard-MIT Math Tournament

Evan Chen

November 24, 2016

Helium is a grading system for math contests. It was developed for HMMT in October of 2016 following inspiration from the Stanford Math Tournament's new Atomic Grader.

Helium is built using Django to be compatible with HMMT's then-existing structure. It features scan-based grading, a rapid-fire score input system, as well as support for direct score entry. Conflict resolution is handled automatically, as well as generating of results according to the HMMT scoring algorithm. The Django admin interface is built in.

This documentation is split into several parts. Chapters are dedicated to graders (day-of volunteers), officers (trusted staff members with extra responsibilities), system administrators (with setup and result checking before the tournament), and finally other developers (who wish to play with code), with each section more technical and detailed than the last.

The appendix consists of a list of terminology which will be used throughout the manual. Also, a note that screenshots may be slightly out of date.

Contents

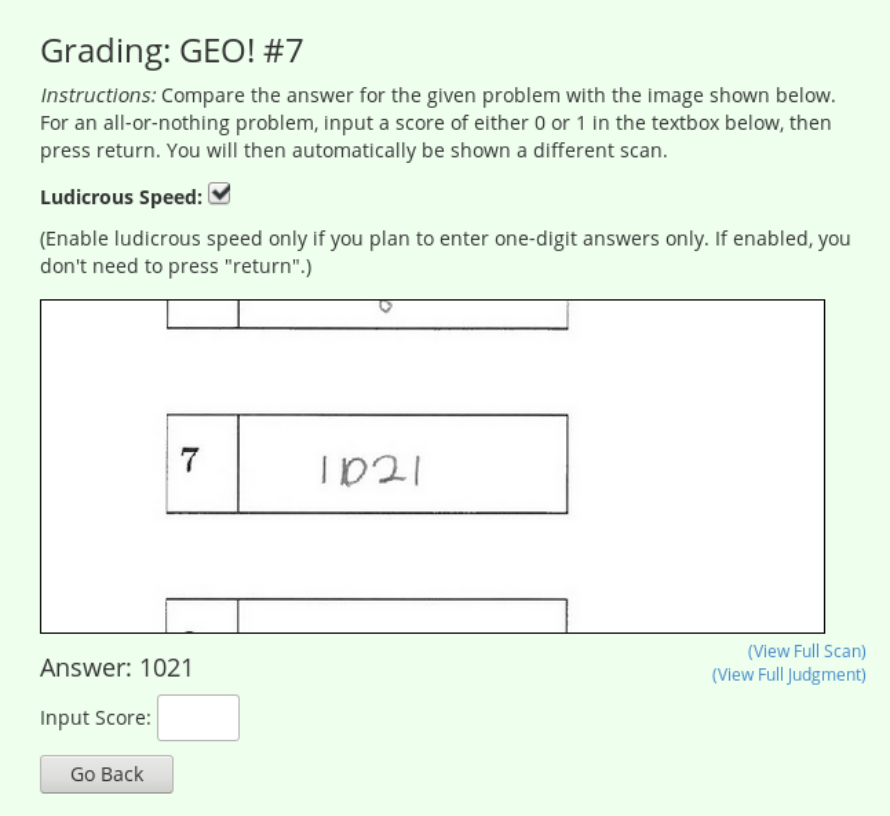
1 Instructions for Graders

This details the tasks which should be performed by day-of graders. Most graders will only need to read this chapter of the manual, and maybe the terminology section (Appendix A).

The two functions that you care about are "Grade Scans" and "Classical Grader"

§1.1 Scan grading

This is pretty self-explanatory.



- Try to select one of the 10 problems from an exam at random, so that we don't end up with everyone problem 1 or something.
- For all-or-nothing problems, type 0 for an incorrect answer and 1 for a correct answer.
- Press "go back" if you made a mistake and want to fix it.

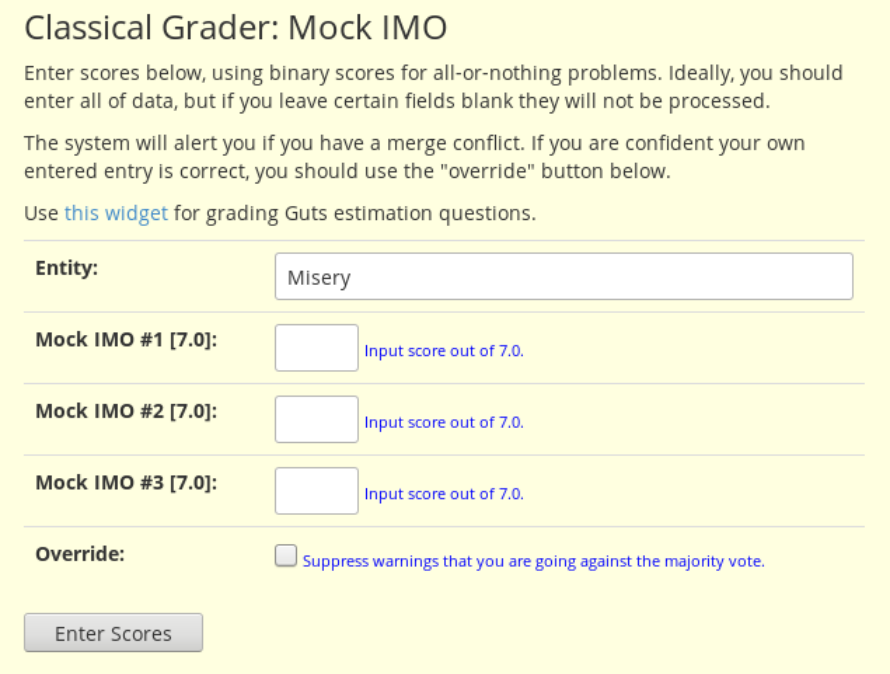
It's possible that you might have some problems where scores entered are not just 0 or 1. That's fine too.

The option "ludicrous speed" is enabled by default (checkbox above), as long as the problem you are grading is not all-or-nothing. This means that you don't have to press enter. If you prefer to take things more slowly, or for some reason the problem you are grading accepts scores other than 0 or 1, you should disable this option.

If configured, the background of the page may be tinted with the color of the test, so that you don't accidentally grade the wrong test.

§1.2 Classical grader

Most importantly: *do not use the classical grader on problems graded by scan*! (The interface disables this by default.)



Use the classical grader for tests which are not graded by scan (for example, the Guts round). There are two ways you can do this:

- Grade by exam, meaning you are grading many problems on that exam.
- Grade by problem, meaning that you are only grading a particular problem (for example a February proof-based team round problem).

Afterwards, this is self-explanatory; select the entity you want, enter their scores, and submit!

This time, Helium will warn you if you are submitting a score that goes against a consensus for that problem (for example if Alice enters 1 and then Bob tries to enter 0 later, Helium will warn Bob). If you are confident you are correct, you can use the override option. However ideally when this happens you should find the person who submitted the wrong answer before and check with them first.

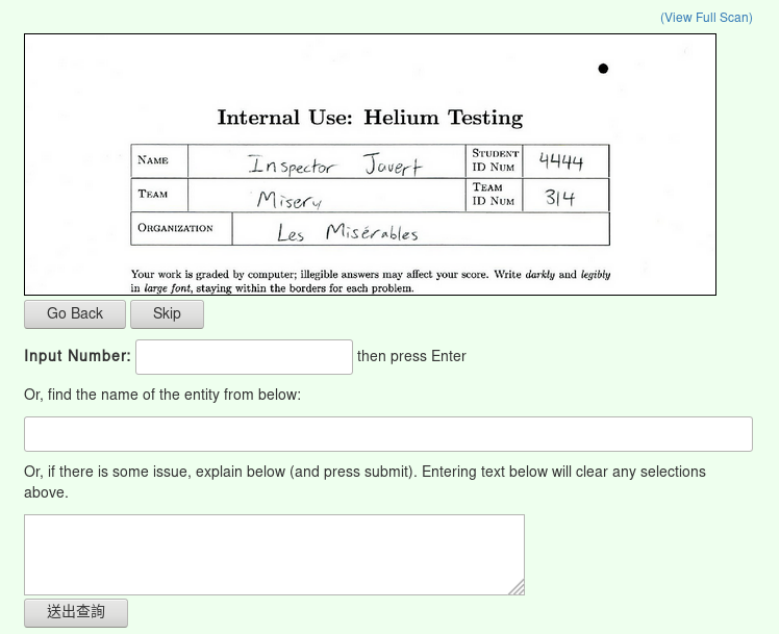
Whenever you select an entity, the scores you previously entered for that entity will be automatically displayed.

If configured, the background of the page may be tinted with the color of the test, so that you don't accidentally grade the wrong test.

§1.3 Matching names

The other option you can do, if you want something a little less thoughtless than pressing the zero or one key, is to help with matching students to names. This feels quite similar

to the scan grader, but the difference is there are three options you can choose for each scan.



There are three things you can do for each student name:

1. Most easily, input the student's ID number. Type in the student's number into the text box; the autocomplete field will then match with the name of the student. Press enter to submit.
2. Alternatively you can try searching by name. Type in the name in the autocomplete field; upon getting a match, select it and press enter. This works equally well. I think it's faster to use ID numbers when possible, but to each their own.
3. If their some other issue with the scan (e.g. there is no name or number at all, or the student doesn't appear to exist), you should flag it for administrative attention. To do this, simply fill in the text box, and press submit. An administrator will then be able to deal with it.

§1.4 Viewing Full Scans

When doing scan grading, you can press "view full page" in order to see the entire scanned page instead of just the cut-out. Just click the "(View Full Scan)" button near the corner of the image.

On this page there is also a switch where you can flag the page for administrative attention, if there is something wrong with it (for example, it is blank, or rotated, or illegible, etc.).

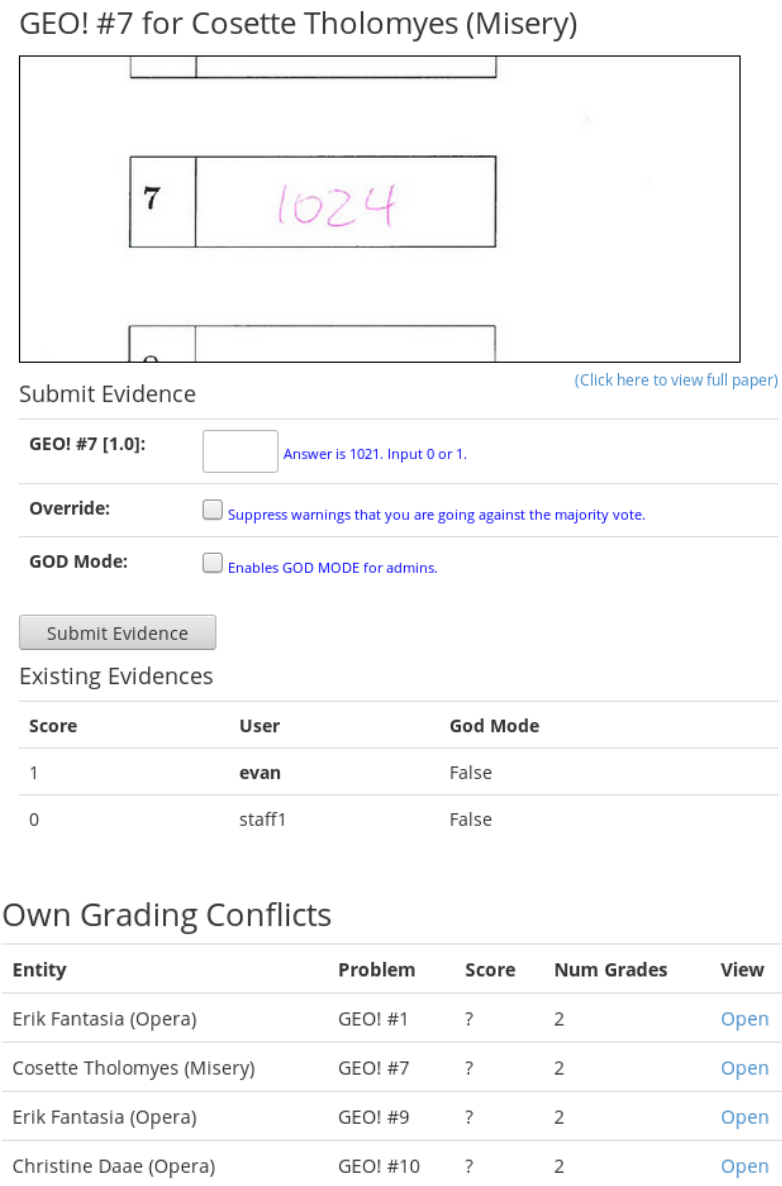
§1.5 Guts round

- Guts round scoring is done using the classical grader.
- For the "estimation problems" you will have to compute the score to assign to the problems at the end. So a "Guts estimation calculator", self-explanatory, has been provided.

- The old Babbage Guts scoreboard should still be working (it now pulls data from Helium).

§1.6 Viewing conflicts

You can see any evidences you submitted which conflicted with other graders by pressing "View Your Grading Conflicts". This allows you to also open any particular verdict, and change your decision on it.



This shouldn't really be necessary for scan grading, because verdicts are settled with a sufficiently large majority vote anyways. That is, if you mis-grade a scan, it will eventually sort itself out.

§1.7 Progress grading

There are some progress reports on how far grading is going for each problem. You can use this if you're not sure which problems needs more help. (Officers may also use this so they can panic as they see things aren't getting done, or something like that.)

Grading Progress

Problem	Done	Pending	Unread	Conflict	Missing
GEO! #1	15	0	0	1	0
GEO! #2	16	0	0	0	0
GEO! #3	16	0	0	0	0
GEO! #4	16	0	0	0	0
GEO! #5	16	0	0	0	0
GEO! #6	16	0	0	0	0
GEO! #7	15	0	0	1	0
GEO! #8	16	0	0	0	0
GEO! #9	15	0	0	1	0
GEO! #10	15	0	0	1	0

§1.8 Sneak peek at results

Some of you may be curious at how teams or students are doing. While I can't endorse this, the interface lets you do so: you can poke around the results page, or look up student papers, etc. If you do look through this, note that:

- Nothing you see is final, and
- All results are confidential.

Note that results are not computed in realtime. The algorithmic scoring always takes a few minutes.

§1.9 Do not use "Upload Scans"

Unless instructed to do so.

2 Instructions for officers

This assumes you have super-user privileges. So it contains some higher-level administration tasks.

§2.1 Uploading scans

Self-explanatory, but some gotchas:

- Pre-process the paper:
  - Remove staples from scans, if any. You may need to have a squad of people do this *en masse*.
  - At the moment we don't have image processing, so please make sure all scans are correctly rotated.
- **Upload the scans in grayscale**, NOT black/white or color. Using color slows down processing, and using black/white makes the scan illegible since we generally print exams on colored paper.
- I strongly recommend using the **following organizational procedure**:
  - Upload the scans in batches of 25 pages each. (Batches longer than this may cause issues on most scanners).
  - Number the batches 1, 2, ..., n. Write the number of the batch in sharpie on the first page, and upload it with the number in its filename.
  - After scanning a batch, staple it and put it aside.
- For safety, uploaded filenames *must* be distinct. (This is to prevent someone from accidentally submitting the form twice, or submitting the same PDF twice.) Filenames must end with either "pdf" or "PDF".
- *Please*, make sure you are uploading the scans to the correct contest! There are some ways to deal with such mistakes but better if they can be avoided in the first place.

When you upload a scan, several problem scribbles, exam scribbles, and verdicts are immediately created corresponding to the items in the PDF file. However since the system does not yet know who is who, the verdicts and problem scribbles will not have an entity attached to them. The "match scans" functionality will then pair them up.

It seems preferable to upload scans in smaller batches, rather than all at once. This way, after the first PDF is uploaded, graders can start marking immediately while future PDF's are uploaded.

§2.2 Marking exams as ready

In the Django administration interface, there are two switches you need to flip for every exam:

- You can mark an exam as "ready to grade", meaning that its names can be matched and its answers graded.
- Separately, you can enable the uploading of scans ("can upload scans").

These are safety features designed to minimize the risk of someone accidentally uploading scans to the wrong test, for example.

§2.3 Dealing with troublesome name matches

There are two ways to get to here:

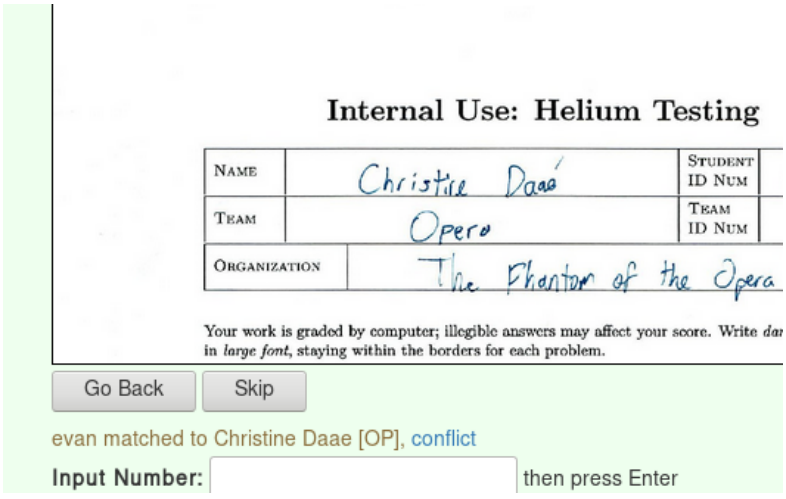
- If you open "find paper", you will get a list of all name matches for which some difficulty appeared.
- If you open the interface for matching exams for X, at the very bottom of the page there is a link to view only the scans which need administrative attention.

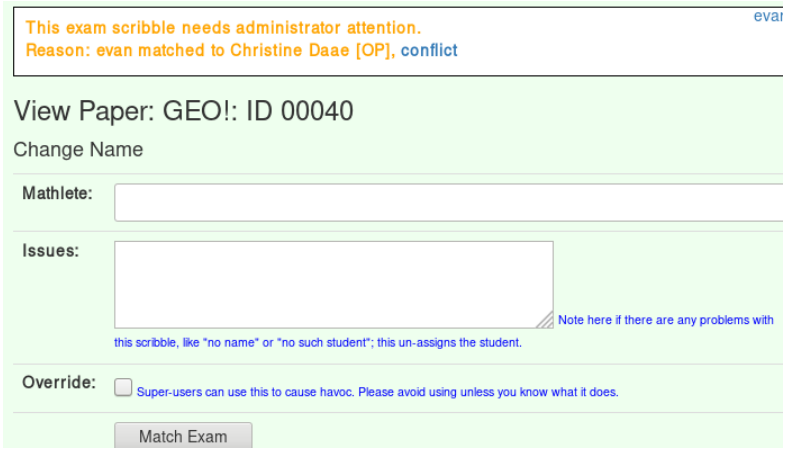
Some of these reasons are entered to humans: for example, "no name". Do with these as you see fit.

The tricky situation is auto-attention generated papers which occur when a student is repeated. Consider the following situation:

> You match a geometry exam to a student "Christine Daae". Later on, you try to match another geometry exam to the same student "Christine Daae".

This is obviously not okay, and the system will complain. An error will appear linking to the first *verdict* it finds for a geometry problem taken by Christine Daae.





At this point you need to fix the error. There are a couple things you could do.

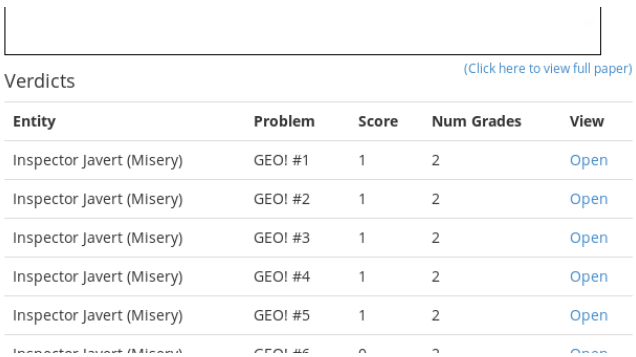
- If you made the mistake this time, then just re-submit the form with the correct name. Easy.
- If you made the mistake the *first* time around, then you need to go back and fix it.

  You can do this by clicking on the verdict provided by the system, and then opening the exam scribble for that verdict. Alternatively, the "Find Papers" interface will let you find the previous paper as well. This will bring you to a match form where you can change the name to the correct one.
- As a last resort, the "Override" switch in the "full scan" page (visible only to super-users) will forcibly assign the paper, and retroactively delete all verdicts and problems scribbles to the contrary. This is a Bad Idea™ and you should only do it if you really understand Helium well and have a compelling reason to do so.

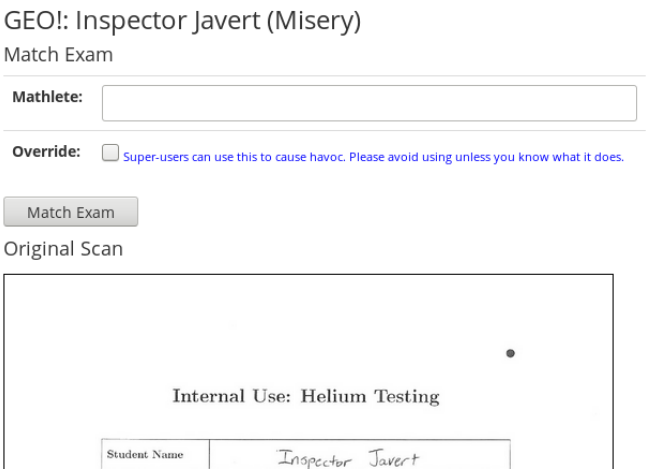
§2.4 Find papers

You can locate specific papers by selecting "Find Papers" from the index page. Some ways to do so:

- You can look up by exam and entity.
- You can scroll through the pages of the PDF files.
- A list of papers which need moderator attention will appear here (see Section 1.4).



If the paper has an exam scribble associated, that will also be displayed, along with a form to change the name associated to it (in case of error).

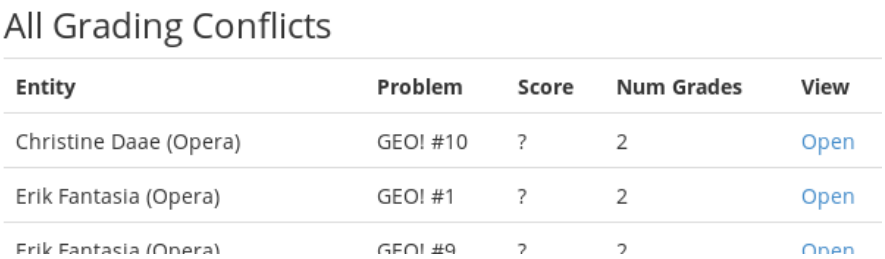


§2.5 Conflict resolution

First, I should describe the logic behind verdicts. For each exam there are two parameters:

- A **minimum majority** for the verdict to be valid, defined as a constant n such that an n-to-1 majority suffices.
- If valid, a **minimum number of graders** needed before the verdict considers itself done. (So setting this number to 2 is double-grading, 3 is triple-grading, etc. and the default is 3.)

Officers get an extra panel called "View All Conflicts". This lets you see the grading conflicts for *everyone*.

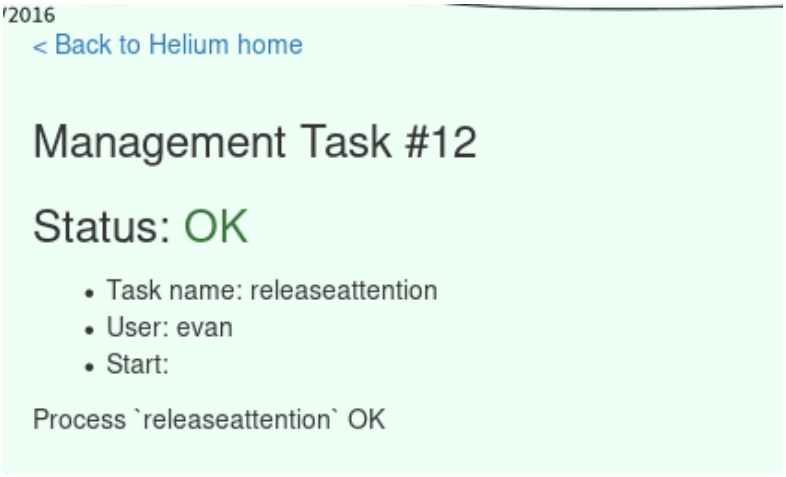


Given a conflict, you can open the verdict page corresponding to it, at which point you can submit your own evidence. If you like being final authority, there should also be an option to submit this evidence in "God mode" which will settle the verdict once and for all.

§2.6 Management

Several commands are available in the `manage.py`. Many of them are revealed through the Helium interface too: on the dashboard, several links to Helium tasks are available.

When you push one of these such links, you'll be directed to page which will auto-refresh once the task is done. This is necessary since the task can potentially take very long (algorithmic scoring).



We'll talk more about these commands in the next chapter.

§2.7 Other Administration

The Django admin interface can be used if there is some operation you need to do not supported by the existing views.

§2.7.1 Someone uploaded PDF to wrong exam!

Well, that's a bummer. Delete the relevant PDF file from the Django admin interface. This will kill of the exam and problem scribbles too.

As of December 2016, deleted problem scribbles will also kill off the verdict to which they were attached. That should mostly save things. No promises though.

§2.7.2 A page is rotated!

If you manage to find a rotated exam, you can first mark the exam scribble as "needing attention". This will block it from being handout out to staff members. Then, you should probably manually grade the exam in God mode.

3 Instructions for Helium administrator

Here are some tasks you need to do to set up Helium to work before the tournament, and after grading.

§3.1 Database maintenance

§3.1.1 Clearing the database

You should first clear out the old exams, verdicts, etc. from the database before another run of the tournament. If you want to save a copy of all the data, you can do so using Django management:

```
python manage.py dumpdata helium --format yaml.
```

This will get you a YAML file with all Helium related data. If you ever need it back you can then reload it with `loaddata`.

§3.1.2 Import exams from Babbage

You need to populate the Helium tables with exams.

To do this, you should go to the problem database and for each contest that you want to export, push the "Export to Helium" link. This will copy all the problems and their answers into Helium, and then open the admin interface so that you can make any manual changes necessary (for example partial marks are disabled by default but you will need to add them into Guts estimation). Remember to input colors corresponding to how the answer sheets will be tinted.

A safety feature is that exam names in Helium must be distinct. So if you decide you want to re-import an exam, you will have to delete the old one first.

§3.1.3 Import entities from registration

In Babbage a command called "Import Entities from Reg" will do this for you. It is careful, and will never import the same entity twice. This is also the "heliumimport" command.

It is very important to **run heliumimport as late as possible**, preferably just after the first individual exam begins. Registration always changes last-minute.

§3.2 Guts estimation problems

First, ensure that the name of the Guts exam is "Guts", or otherwise maul `babbage/views.py` to make this work.

Next, if there are any estimation problems the scoring functions for these need to be added. You can do this in the admin interface, under "Guts Score Funcs". This should be entirely self-explanatory.

§3.3 Printing and checking scans

You should make sure that a large number of answer sheets are printed, and in different colors between tests to avoid errors.

Unfortunately, right now the scan regions (i.e. the cut-outs) are hard-coded into `scanimage.py`. So before the tournament you should check the scanners being used and verify that the cut-out regions are compatible with the scanner.

The file `scanimage.py` has a facility to make this easier. If you run

```
python /path/to/scanimage.py FILENAME.pdf
```

then it will generate (in the current directory) the images corresponding to the cut-outs. Thus you can test this locally and make adjustments to the regions if needed.

§3.4 Name stickers

Make sure tournament directors are printing out name stickers for the students; this will make grading life a lot easier. Also, on personalized schedules for teams, print out ID numbers. Doing so will expedite the process of grading significantly.

This task takes place outside of Helium, and varies between tournaments. The bottom line is that **ensure students and teams know their ID's**.

§3.5 Sanity checking results

Once grading is done, you should do the following clean-up tasks.

- First, open the Verdict table in Django admin and look for verdicts which are "inaccessible". In theory there should be none. If such a verdict exists, do a sanity check to make sure it's an anomaly, then delete these verdicts.
- If there are verdicts with no names, verify this is correct: these should only come about from no-name papers, or papers yet to be matched.

  In November 2016 for unknown reasons, some exam scribbles got matched but the underlying verdicts for them were left as None. You can auto-magically fix these by running the management command "update scribbles" which will fix any issues.
- Look in the Entity table for any entities which have no verdicts attached to them. First delete all individuals with no verdicts: this probably means that they did not show up.

  Teams can also not show up, but they can also leave early and not take Team or Guts round. That's why it's necessary to delete no-show individuals first. Only after this, if a team has no verdicts *and* no individuals, then one can delete it.

§3.6 Last grading commands

There are two different management commands at play. You need to run both, in this order.

- "algscore": this command is slow, taking about two minutes per run. It computes the α and β values for each student based on the set of observations.

- "grade": This command is fairly fast, it should take no more than a few seconds per run-through. It collates together all the results into a table which are then displayed in results reports. This is necessary because otherwise the process of aggregating the scores is too slow.

Run both commands, then look through the reports to make sure everything is okay.

It may be wise to run "grade" throughout the day, just to make sure that things are looking okay. The α and β values won't be correct, but that's okay; we don't need those values when we're just sanity checking results.

4 Notes for software team

This contains low-level technical notes on some of the non-obvious moving parts behind the whole system. This is intended to help you extend Helium in some way.

Most of the code I tried to comment or `help_text` well. But if anything is still confusing, that's probably my fault. Feel free to contact me at `chen.evan6@gmail.com`.

§4.1 Personal request

I have a personal request before you start editing, which is that my name remains intact in `LICENSE.txt`, this documentation, and other places (for example file headers). You are free of course to add your own name should you make substantial contributions.

The reason I ask this is because this system represents well over 200 volunteered hours of my free time pooled into over 400 commits. I built the entire system from scratch with no assistance from anyone else. Since I didn't collect any payment for this work, the least I ask is that I am recognized for it.

Thanks!

§4.2 A Warning

I found out the hard way that a typical tournament has maybe $N = 30000$ verdicts in it. This means you should make sure your code does not do either of the following per call:

- Do not use algorithms that run in $O(N^2)$ time.
- Do not make $O(N)$ queries to the database.

For queries grabbing the entire verdicts table (or large portions of it), it is almost always better to use the `queryset.values` method. For these large values of N, creating N instances from the table is just too slow.

§4.3 Test data

§4.3.1 Location

Some test data has been provided for you. You can look for it in the following places:

- `:/helium/static/batchX-samplescans.pdf`
- `:/helium/static/batchY-samplescans.pdf`
- `:/helium/fixtures/*`

The easiest thing to do is to load the two fixtures in the "setup" directory, which will populate the exams and entities tables. The sample scans correspond exactly to the entities in these fixtures.

You can also load fixtures in the `:/helium/fixtures/scenarios` directly which are snapshots of the entire database at various points in time in the test run (after matching exams, after grading, after algorithmic scoring, etc.). But note that the `/media/` folder is NOT included.

Warning: the scenarios folder contains a superuser with name and password "evan" as well as a staff user with name and password "staff1". Consequently, be careful to NOT run these on production, as we will then have some very insecure user accounts!

§4.3.2 Synopsis of test data

Synopsis of test data:

- There is a single scan-based exam, called "GEO!", and there is a single non-scanned exam, called "Mock IMO". The former is 10 problems scored algorithmically. The latter is 3 problems worth 7 points each.
- There are 16 mathletes split into three teams of five, corresponding to characters from *Lés Miserables*, *Wicked*, and the *Phantom of the Opera*, plus one unaffiliated individual named "Evan Chen".
- The two sample batches contain all their "GEO!" exams. The answers to that exam corresponding exactly to the real answers to the HMMT February 2016 Geometry test.
- Under the registration folder are some fixtures for testing import to registration and so on.
- Under the problems folder are some fixtures for loading some example data into the problem database.

§4.4 Moving parts

When possible I tried to keep everything in Helium self-contained, not relying on even files outside the directory.

§4.4.1 Helium dependencies

Here is a partial list of things externally that Helium relies on; it may be incomplete. All this is at time of writing; other people move things around all the time so God knows if we will even have `core/settings.py` in a few years.

- `requirements.txt` in the root directory has some Helium specific additions. In particular `wand` which does image processing in Python. The command

  ```
  pip install -r requirements.txt
  ```

  should do the trick.
- Image installation is done through the EB extension `01yum`. This installs `imagemagick`, `freetype`, `ghostscript`, which is used for the image processing.
- Helium needs to host its media on an external S3 server, because scan images need to be distributed. So currently I have an S3 bucket called `heliummedia`. The connection is in `core/settings.py`, The S3 bucket itself lives inside AWS along with the entire HMMT application.
- `core/settings.py` should also have some basic applications installed, such as
  - 'django.contrib.admin',

- 'django.contrib.auth',
- 'django.contrib.sessions',
- 'django.contrib.messages',
- 'django.contrib.staticfiles',
- 'storages'.

§4.4.2 Things depending on Helium

- The Guts scoreboard reads its data from Helium. So if you make changes to Helium, check there too.
- Registration import into Helium, as described before.

§4.4.3 Other nice things to know

Some not terribly Helium-specific things, but you should still know:

- Logging configuration (not specific to Helium, but nice!) is done by the EB extension `00logging` and configured in `core/settings.py`.
- Not Helium-specific, but the EB extension `05managepy` is responsible for running `collectstatic` and `migrate`.

§4.5 Git subtree

There is an upstream URL which publishes Helium-specific code. This way other math tournaments can use the same code themselves.

This upstream URL is maintained by a subtree via the command

```
git subtree split --prefix=helium -b helium
```

to create a branch `helium` containing only code from this folder. Thereafter, we push this branch into the Git helium repository (to the master branch on Git: note that the branch names differ).

The `README.txt` files provides a brief documentation on how to get set up with Helium in a different Django project.

§4.6 A wishlist

Some possible to-do things:

- Image processing: auto align scan regions and cut-outs.
- Proof grading (long shot).

A Terminology

The grading system is itself known by the name **Helium**. Each event will be referred to as a **contest**.

§A.1 Participants

In each contest there are several participants. The students are referred to as **mathletes**. These students are often grouped into **teams**. A mathlete may have no team; in this case we say they are an **unaffiliated individual**, or sometimes just "individual" for short although this is confusing.

An **entity**[1] refers to someone, either a mathlete or a team, who takes an exam.

A **user** refers to a staff member of the contest (not a student or coach). A **superuser** refers to a higher-level administrator of the contest; for example a head grader, or problem czar, or tournament director, or software director. To be more exact this is the set of users with administrative privileges on the Django website.

§A.2 Exams and problems

A contest consists of several **exams**[2], each of which has several **problems**, which are numbered 1, 2, ..., n in each exam. These exams are **taken** by entities, who must attempt to solve the problems on the exam.

A problem has the following properties:

- The problem has a **weight**, which is the number of points a contestant earns for solving it.
- The problem may allow **partial** marks, or it may be an "all-or-nothing" problem.

Exams may be "individual" or "team" exams, depending on which types of entities take them.

§A.3 Scans

A **problem scribble** refers to the scan of a contestant's answer to a problem. An **exam scribble** refers to the scan of a contestant's entire answer sheet for an exam. You won't see these two terms much in the user interface, but they are used internally in the source code just about everywhere.

A **paper** refers to the set of answers for a contestant to any given exam, regardless of whether the paper was fed into a scanner or not.

§A.4 Grading

Suppose an entity E attempts a problem P, Helium then needs to give a **verdict** to the pair (E, P), which consists of a score assigned to the pair (E, P). This represents the judgment handed down by Helium to E for this problem P.

[1] This abstraction is useful because an exam doesn't care so much who takes it.

[2] Helium deliberately avoids the word "test", since this may be confused with "testing" and so on

To generate this verdict, one or more users will input a score for the pair (E, P). Each such score by a grader is termed an **evidence** contributing towards the verdict, and we say the user **submits** this evidence. The verdict is **valid** if there is a clear consensus what the final score should be (by default this is at least a 3-to-1 majority, but this may be adjusted per exam). The verdict is **done** once it is valid and there is enough evidence.

There is one possible exception: super-users may submit a evidence in **God mode**. When this occurs, it overrides any and all evidence submitted by regular users.

§A.5 Algorithmic scoring

At HMMT a complex algorithm is used for assigning the weights to the individual tests. Details of this algorithm are available at

`https://www.hmmt.co/static/scoring-algorithm.pdf`

A copy of this algorithm is also included in an appendix at the end of this document.

Let $\mathcal{A}$ be the set of exams scored using this algorithm. Each problem on an exam in $\mathcal{A}$ obtains a weight which is called a **beta value**, denoted $3 \le \beta \le 8$. Moreover, each entity taking one or more exams in $\mathcal{A}$ is then assigned an **alpha value**, which represents their strength, and is denoted $\alpha \in \mathbb{R}_{\ge 0}$. These α and β values are stored in database, since they are quite hard to compute.

At HMMT the alpha values are used to aggregate individual scores across all tests, and thus used to rank individuals.

B HMMT Algorithmic Scoring Details

§B.1 Synopsis

§B.1.1 Input

Each exam contains at least one **problem**. The exams are taken by several **contestants**.

This algorithm takes several **observations**: i.e. it takes as inputs pairs (c, p) of contestants and problems, and for each such pair is told whether or not c **solved** p. During the actual tournament, every problem is attempted by every contestant; however, the algorithm will not use this assumption.

§B.1.2 Output

Based on this, the algorithm outputs:

- An **strength** $\alpha_c \in \mathbb{R}_{\geq 0}$ for each competitor c, and
- A **weight** $\beta_p \in [3, 8]$ for each problem p, which represents its difficulty.

Note that the strength is *not* the same as the score of a contestant. In particular, as described below, each contestant is given a single strength value across all exams.

§B.1.3 Usage in contest

The calculation at the actual HMMT is done in one pass. For example, at HMMT February 2016, there were three exams with 10 problems each and 700 contestants. Thus there were $700 \cdot 10 \cdot 3 = 21000$ observations as input. Note that the algorithm is applied only once! For example the observations on the Algebra test affect the weights of the Geometry test.

For each particular 10-problem **exam**, the **score** of a contestant on that exam is the sum of the difficulties β_p for each problem that they solve. The score of a contestant is used to determine their rankings within each individual test. Scores across different exams are not comparable.

For individual sweepstakes, the strength α_c is used instead (in order to ensure that no particular exam dominates in determining the aggregate individual score). Contestants are ranked based on the value of α_c assigned to them. When determining team sweepstakes, the individual contribution (out of 800) is proportional to the sum of α_c across the team members c, scaled so that the strongest teams earns the maximal 800 points.

§B.2 Design

§B.2.1 Advantages

The scoring algorithm is designed to meet the following criterion:

- The system provides a careful way to **compare scores across tests**, leading to less noise in the aggregate individual and team rankings.
- The system **factors all possible inputs**, rather than for example merely the top 10 contestants or otherwise, as occurred in previous HMMT tournaments.

- The system is **resistant to preconceived difficulty**. Originally, problem czars were forced to estimate the difficulty of every problem by assigning it a weight; this leads to possible noise in the results. The algorithmic system determines the difficulty based on the actual performance.
- The system is **hard to exploit**, in part because it takes inputs from all contestants, and in part because it is so complicated that a contestant is probably better off thinking about the exam problems.

§B.2.2 Qualitative criteria

The following additional criteria are satisfied:

- The strength of each contestant is nonnegative,
- The strength of a contestant is unbounded, but not infinite even if they solve every problem.
- The strength of a contestant is zero if they solve no problems.
- Problem weights lie in the interval $(3, 8)$.
- The distribution functions are smooth.

§B.2.3 Choice of priors

In light of this, we select the following parameters, which describe how the α_c and β_p should be interpreted.

- The strength $\alpha = \alpha_c$ of each contestant c is *a priori* distributed in $\mathbb{R}_{\geq 0}$ according to
$$\mathbf{P}(\alpha) = \exp(-\alpha). \tag{B.1}$$
- The weight (difficulty) $\beta = \beta_p$ of each problem p is *a priori* distributed in $[3, 8]$ according to
$$\mathbf{P}(\beta) = \exp\left(\frac{5}{(\beta-3)(8-\beta)}\right). \tag{B.2}$$

Now we relate the strength $\alpha = \alpha_c$ of contestant c to the difficulty $\beta = \beta_p$ of problem p by postulating that the probability that c solves p is

$$\mathbf{P}\left(c \text{ solves } p \mid \alpha_c = \alpha \text{ and } \beta_p = \beta\right) = \frac{\exp(-\beta/\alpha)}{1+\exp(-\beta/\alpha)}. \tag{B.3}$$

Assuming (B.1), (B.2), (B.3) and holding fixed the set of observations of the tournament:

> *The algorithm outputs the unique set of α_c and β_p for which the outcome of the tournament was most likely.*

The rest of the document describes how to actually compute the maximum.

§B.3 Algorithm Description

The algorithm is a slightly modified version of the Rasch model.

§B.3.1 Pseudocode

Let us for brevity denote the function in (B.3) by

$$w(\alpha, \beta) \stackrel{\text{def}}{=} \frac{\exp(-\beta/\alpha)}{1+\exp(-\beta/\alpha)}.$$

Note that $w(\alpha, \beta) \in (0, 1)$,

Now consider the following systems of equations, whose origin we explain below. First, for every $\alpha = \alpha_c$ we have the equation

$$0 = -1 + \sum_{p \text{ solved by } c} \beta_p \alpha^{-2} - \sum_{p \text{ took by } c} \beta_p \alpha^{-2} w(\alpha, \beta_p). \tag{B.4}$$

Moreover, for every $\beta = \beta_p$ we have the equation

$$0 = \frac{1}{(\beta-3)^2} - \frac{1}{(8-\beta)^2} + \sum_{c \text{ taken by } p} \alpha_c^{-1} w(\alpha_c, \beta) - \sum_{c \text{ solved by } p} \alpha_c^{-1}. \tag{B.5}$$

The algorithm binary searches for a set of $\vec{\alpha}$ and $\vec{\beta}$ which simultaneously satisfy (B.4) and (B.5). It is proved below that this set of parameters is unique; this is the output of the algorithm.

§B.3.2 Derivation

We show the derivation of equations (B.4) and (B.5).

Fix a set of observations for the tournament. Now consider a choice of $\vec{\alpha}$ and $\vec{\beta}$ is known. Then the probability of the observations is distributed according to

$$F(\vec{\alpha}, \vec{\beta}) = \prod_c \mathbf{P}(\alpha_c) \prod_p \mathbf{P}(\beta_p) \prod_{c \text{ solved } p} w(\alpha_c, \beta_p) \prod_{c \text{ missed } p} (1 - w(\alpha_c, \beta_p)) \tag{B.6}$$

assuming the events are independent.

We now claim that:

Proposition B.3.1

If a choice of $\vec{\alpha}$ and $\vec{\beta}$ optimizes F, then it is a solution to (B.4) and (B.5). In other words, the outputted $\vec{\alpha}$ and $\vec{\beta}$ are those for which the observed outcome was most probable.

Proof. Note that

$$\prod_{c \text{ solved } p} w(\alpha_c, \beta_p) \prod_{c \text{ missed } p} (1 - w(\alpha_c, \beta_p)) = \prod_{c \text{ solved } p} e^{-\beta_p/\alpha_c} \prod_{c \text{ took } p} \frac{1}{1+e^{-\beta_p/\alpha_c}}.$$

From this, and plugging in (B.1) and (B.2), we get

$$F(\vec{\alpha}, \vec{\beta}) = \prod_c e^{-\alpha_c} \prod_p e^{-\frac{5}{(8-\beta_p)(\beta_p-3)}} \prod_{c \text{ solved } p} e^{-\beta_p/\alpha_c} \prod_{c \text{ took } p} \frac{1}{1+e^{-\beta_p/\alpha_c}}.$$

It is equivalent to maximize $\log F$, which is

$$\log F = -\sum_c \alpha_c - \sum_p \frac{5}{(8-\beta_p)(\beta_p-3)} - \sum_{c \text{ solved } p} \frac{\beta_p}{\alpha_c} + \sum_{c \text{ took } p} \log \frac{1}{1+e^{-\beta_p/\alpha_c}}.$$

Then right-hand sides of (B.4) and (B.5) are the *partial derivatives* of $\log F$ with respect to α_c and β_p for each c and p. The partial derivatives equal zero at any local maximum of $\log F$, as desired. □

In fact, one can actually check that the right-hand sides of (B.4) and (B.5) are monotonic in α and β respectively. Consequently, F is convex. This implies:

Proposition B.3.2

The function $\log F$ has a unique point at which all partial derivatives vanish, and that point is a maximum for F.

This also implies that one can numerically solve the equations by the following binary search procedure. We let $\vec{\alpha}_0$ be an arbitrary point. Then we let $\vec{\beta}_n$ be the solution to (B.5) given $\vec{\alpha}_{n-1}$. Notice that one can actually solve for each component β_p using binary search separately, as each equation of the form (B.5) involves only a single β_p. Similarly, we can let $\vec{\alpha}_n$ be the solution to (B.4) given $\vec{\beta}_{n-1}$ in the same fashion. The pairs $(\vec{\alpha}_n, \vec{\beta}_n)$ will converge to the maximum since we are looking at partial derivatives of a concave function, so we now have an iterative procedure for computing the output.