Automating Role Recognition in Social Graphs

Project Proposal—CS 5350

Alex Clemmer

September 5, 2011

Summary

1 Methodology

The main bottleneck in our project was clearly the lack of available resources: almost everything so far has been done essentially by one person— one person to do the programming, the planning, the coordination, and to communicate the state of the project to relevant authorities.

For this reason, we thought the healthiest approach would be to approach the project as a series of interesting questions about outreach. At least for the purposes at hand, it seems more useful that we get a large number of noisy results for a variety of questions than it is that we get a certain result for one question. This has the benefit of demonstrating what options outreach programs like ours have when they seek answers to similar questions. Additionally, we hope that this will stimulate discussion about what sorts of things are important in our current outreach regimen.

I've here selected the questions whose answers are most relevant to the problems I've personally encountered in my tenure as volunteer and employee in both of these organizations.

1.1 What does it take to get kids to visit a website?

Getting kids to voluntarily re-engage (*e.g.*, send mail to the person doing the outreach) after an outreach event is a notoriously difficult problem. The most interesting question for us was what the limits of re-engagement are, and the first way we began to explore this problem was to explore what it takes, specifically, to get kids to just visit a website.

The high-level seed model for this plan was to collect email addresses volunteered by the students, and then to send out emails to the students with a link to that website included. We were able to use some artefacts of STEP's outreach research, most notably their survey cards, which explicitly ask for email addresses, and which almost all students provide. By doing what STEP used to do anyway by default, we actually ended up collecting a lot of useful information (*e.g.* race, sex, email).

Actually implementing the website was significantly trickier. We'll detail the technical aspects of this in the "Technical Details" section, as they're both less important, and easier to gloss through, than the other details we can cover here. That said, the 10,000-foot view is that this challenge had three major components: the website needs to be accessible *only* to the people we send emails to (so that we avoid pollution of traffic statistics); the website needs to be able to send out emails (*e.g.*, registration and password-forgotten emails) automatically; the website needs to be able to aggregate usage statistics effectively.

The first problem was solved by implementing login functionality. This, along with email generation and statistics aggregation were implemented almost entirely by hand.

This entire system became production-ready by about early- or mid-march. It has since then been used primarily in CS outreach presentations. When I couldn't be there specifically for the presentations, people would retrieve the cards anyway, and then give them to me, and I would enter them into the system.

The priority here is statistics aggregation, which also turns out to require the most amount of and work. In general it is difficult to generate statistics based on user behavior, and the barrier is almost always what you can derive from information given to you by the technology itself. One example of this is our attempt to aggregate the total number of people who opened our emails versus the total number of people who actually followed the link. Our idea was that, by including pictures hosted on our server in the email, we could determine roughly how many people opened emails based on how many times the image was requested, which in turn should give us a good idea of how many people opened our email versus how many people followed our link.

This, like a lot of the statistics aggregation, turned out to be less simple than we thought: some email services (*e.g.*, gmail) block images by default, and where this is not the case, finding which users were unique turns out to be *quite* difficult (we often got twice the number of image requests than the actual number of emails we sent). We suspect that this will be a significant focus for any system that hopes to implement this methodology on the large scale.

Incidentally, the email we sent out reads, verbatim:

> Hey there `@firstname`!
>
> This is `@rand_team_member`, I vsited your class and gave a presentation about computer science. I just wanted to take a second or two to let you know that, if you have any questions at all, you can definitely email me about them. I took this job because I like talking to people about Computer Science.
>
> Also, if you're interested we're building a website as part of this school-visiting project, and we were sort of hoping you'd have a look:
>
> `@link`
>
> Again, I hope you had as much fun as we did, and if you have any questions, please ask away.
>
> `@rand_team_member`

You may notice that the first line has a typo. I actually put that in on purpose because I read a white paper about customer service that demonstrates that auto-generating emails with misspellings causes people to rate your service higher, usually because they believe that you wrote it just for them.

1.2 When kids do re-engage, what content are they most interested in?

There were initially three propositions here: Karen Krapcho, who locally administers the STEP grant, thought that interactive content (*e.g.*, a video or a game) would be the most popular content. Deidre Schoenfeld, who heads the EA program, thought that it would be randomly-selected science links. I thought it would be a link to the Facebook profile of an engineer who actually visited the class.

We implemented all three of these on the home page (which is available just after you log in). The random science link is at the top, just under the navbar, the interactive content (usually a video) is to the left, and a link to the Facebook profile of a random engineering student who visited your school is on the right.

Again, we'll skip the bulk of the technology problems here, but I will say that the difficult part of this was developing robust methods for determining how much kids were interested in one bit of content versus another. The most useful statistic for us was the time between page requests, and the number of people who clicked on the Facebook link.

One example of the sorts of things we looked at was how long it takes users to click on one of the three links, and how long it takes them to issue another page request for our website. If they take a long time to come back (or if they never come back), then the content we've provided them with is probably valuable to them. If they click on a link quickly, it means that this content is especially important to them, and it was obviously more important than other things on the page.

Before I get into how our content was selected, I would like to emphasize that what we've learned here is not that a certain type of content *can't* be popular, but rather, that there is some content that is *very* popular.

That said, the random science links were generally just maintained by me specifically. I got almost all them from content-rated news sites like Reddit. My selection process could have

been refined, – I basically only picked computer science links that I thought high school kids could understand, and that were highly-rated in these communities – but as we will see, one of the types of content outstripped the other two so significantly that I just didn't spend time on developing this point.

The interactive content was initially envisioned as an embedded game of some sort, but lacking both time, resources, and expertise, I simply added embedded videos, usually from YouTube. They tended to be gee-whiz style science videos that we know from experience presenting tend to be popular. Future projects could probably capitalize from the fact that our EAE program builds Flash games, which may actually turn out to be really popular.

The Facebook links were straightforward: we pulled profiles directly out of their public Graph API. Important to note is that initially these profiles led to the actual profiles of the engineering students, but Dr Furse helpfully pointed out that by providing stub profiles communally available to everyone on the team, we can minimize the chances of inappropriate contact between the engineers and students. We thought this was good advice and implemented it. The links themselves look pretty much like the standard Facebook profile badges—they have profile pictures, a name, and the major (or concentration) that that person is listed under.

2 Results

2.1 What does it take to get kids to visit a website?

By far the most significant factor in the results we got (*i.e*, the number of hits we got) was the amount of time elapsed since the presentation. If we acquired the information more than a few weeks ago, as little as 5% of a class would follow the link through. When we sent the email out the same day as the presentation, the response in general was pretty high: usually between 20 and 40%:

Emails sent out	**Repsonse %**	**Time elapsed**
35	5.7%	3 weeks
42	9.2%	2 weeks
32	25.0%	1 day
43	30.2%	1 day

There are admittedly other things at work here too—over time the system was improved, which you will see in the next section. But among the changes we made to increase the click-through rate, this trend stands alone: dragging your feet kills response. To confirm this suspicion, I split two of our visits into halves, and emailed them at separate times:

	Emails sent out	**Response %**	**Time elapsed**
Class 1a	22	20.8%	1 day
Class 1b	23	13.0%	1 week
Class 2a	25	24.0%	1 day
Class 2b	22	13.6%	1 week

It would have been interesting to see how this change manifests across different types of classroom (*e.g.*, a math classrom), but this is not actually something that the STEP team picks—they just sort of accept any classroom that will have them. Until we have at least another year of data about this, it will be hard to establish trends based on that metric alone.

We should also say that, at this point, we were not yet recording demographic data like age or sex. Even so, while we can't make any strong statements about why kids are visiting the site (or not) based on this data alone, this is the experiment was the first solid indicator that this could be done. It also told us that success depends on *not* being sluggish about getting the email out.

2.2 When kids do re-engage, what content are they most interested in?

The results given the content scheme describe above are pretty clear—the kids are interested in the Facebook profiles. In fact, it's not really close—of the people who clicked on a link (about 20%):

Content type	**% total of respondents**
Random Science Link	5.8%
YouTube Video	9.2%
Facebook link	85.0%

One important thing to note here is that this may in part be our choice of content. May be it's true that there are some Random Science links that are better than others. Maybe games are better than YouTube videos.

What we can conclude is that there is at least a passing interest in the engineers as people, and it thus may be useful to pursue outreach at a more personal level than to simply show up at classrooms, talk, and leave forever.

In general, it's also sort of hard to believe that this is a superficial trend. If true, it further corroborates what intuitively seems like it must be the case anyway: students are not only interested in the subject, but also in the people themselves. In designing new outreach programs, it would seem ill-advised to ignore that something is going on here.

There's another point of interest here, too: not all of our team members are equally good at outreach. In particular, one of our team mates, Shandice Beal, seems to just be *better* at it than the rest of us, and in particular, better than the white males of the group, myself included.

We tell this by taking the average time it takes a user to click on the Facebook link, and then to compare the average time it takes for a user to respond to the Facebook link *one particular user*. In the following table, a negative value indicates that the response time for a particular member is an *improvement* over the average, where a positive value indicates the opposite.

Team member	**% increase in response time**
Shandice	-28%
Sarah	-26%
Morgan	+12%
Landon	+16%
Alex	+15%

Also important to note is that Shandice and Sarah were not at every presentation, and yet their averages are very high. It may or may not be incidental, but it's worth noting that Sarah is Asian, and Shandice is African-American, where the rest of us are white and male. This may be an incidental point, but when we do visits or tabling events, I can say (anecdotally) that the kids, and in particular, females and minorities will almost always approach Sarah and Shandice. It's worth at least thinking about.

3 Next Steps

In the interests of brevity, we're going to omit the smaller conclusions we've come to. The results we've just discussed are, in our opinion, the take-home messages of the semester's work.

One interesting impact of having a database full of names, demographics, and data about interests, is that this could potentially be a useful sustained outreach tool. It is attractive for us to think eventually about finding what people are strongly interested in, and then to build events around those things. This addresses a problem we have felt strongly about for a long

time: in our opinion, impact on kids is derived mainly from *sustained, effective* contact with outreach programs. We would like to continue implementing this, and to see it implemented as such.

Additionally, it would have been interesting to explore in more detail which content users are interested in. In particular, it would have been interesting to explore this across several demographic boundaries, especially race and sex. We'd like to see the outreach program more targeted towards reaching people who wouldn't otherwise consider engineering, and we think this is probably the best next step to accomplishing that goal.

Much of the final work done on this project has been to position it to help deal with these problems. In the end, we think that computer science is uniquely positioned to offer solutions to problems like this, and even though it's not surprising that something like this is not currently in place, we hope that people will agree that it is a reasonable direction to pursue. And especially if they're already paying for it anyway.

4 Technical Overview

4.1 Backend web technology

You can find the website at http://ineering.com. This somewhat cryptic URL is mainly for historical reasons: I would have liked to make it http://eng.ineering.com, but for reasons I can't remember, was told no. It would have required additional funds, so maybe that's it.

The website is written up and down with Ruby, and in particular, with the Rails framework. There are a couple of interesting technologies that I think are worth talking about:

4.1.1 Authentication scheme

We use the authentication plugin `authlogic`, which generally made a lot of authentication problems easier. In particular, it was useful because it generated the database schema and the Rails models that handle them without any fuss whatsoever. As long as your controller calls the right methods when you get a POST from a login form, you also get a lot of "free" fields, like login count, last login time, and so on.

Password authentication is implemented by a multiple-stretched SHA 512, seeded by Mersenne twister, both of which are provided. Users log in via email. The cookie is persistent, but not long-term.

We have chosen to default every page on the site to redirect anonymous users to the login page—you must log in to do anything. Additionally, there are 2 tiers of users: the students and the admins. Again, pages are available as part of a white list—by default, everything is available to admins only. The only pages available to standard users are the pages we explicitly white list. Currently we've white-listed the home page and the page that lists the engineers who came to your school. But other things, like the link-submit page, are not available to standard users.

Additionally, it bears mentioning that we don't have a page that allows even admins to view the list of all the students. This is a security issue—we want to explicitly discourage any inappropriate contact with the students. Admins have access only to pages that aggregate statistics. Information about students is only available at the root database level.

Registration is also disallowed for anyone except admins. Registration can be done individually, via a registration form, or *en masse* via a form with a variable number of registration fields. When you register someone, they get the automated email described in the first section. The link includes a perishable token at the end of the URL that logs users in by default; since Rails has probably the best routing infrastructure of any web architecture, this was fairly easy to implement. As soon as users click on this link, they are immediately logged in and asked to change their password.

4.1.2 Email interface

One particularly critical feature here, like any web application, is the email interface. We used gmail's SMTP API, which is simple in itself, and which Rails makes about as simple as it can be. Still, this was probably the hardest part of the project.

The main issue we ran into was not getting our computer to send out the mail request, but to get the Heroku server to do this. Shortly after we went through the (awful) process of figuring this out, they actually released a Heroku plugin that does this automatically, and then forced us to switch. Heroku is an unequivocally great service, and second to absolutely none for web hosting, but this was annoying.

Emails in Rails are handled just like view renders for page requests. The difference is that it sends this document (HTML, text, whatever) out to the mail server, rather than to the browser.

Gmail caps you at 100 messages a day unless you pay. This is fair, but it did occasionally cause annoying problems, particularly since the email server did not send back errors when the mail wasn't sent. To be fair, though, we didn't handle them explicitly either, so we wouldn't have known that they were sent out unless I was looking at the logs anyway.

4.1.3 Hosting and web services management

Pushing and deploying a Rails app is incredibly simple: `heroku create && git push heroku master`. Git is a distributed version control system that allows you to track, control, and merge between, an arbitrary number of remote repositories. Deploying any Rails app is a matter of using this DVCS to push to the Heroku server. The command above creates the remote repo, adds Heroku's server as a remote, and then pushes your project to the remote server. The server launches it automatically.

Setting up DNS and the like was about as simple as someone with no experience whatsoever could have expected; initially I tried to do it manually, and it didn't work, but when I used their custom plugin, it magically began to work.

In Rails, all database work is actually handled transparently by Rails itself. Users specify schema by Rails models. So although we use Postgres, strictly speaking, this was arbitrary; actually, we use Postgres mostly because Heroku does.

Rails also makes AJAX pretty transparent; the Facebook stub profiles are actually populated on the page asynchronously. I'd actually forgotten that intense AJAX is usually not painless until I had to do it again as an assignment for class.

4.1.4 Testing

We were going to use Cucumber, but ended up just using the baked-in testing framework provided by Rails. By religious dictum, Rails actually has unit tests built in for every model, view, and controller that you generate, provided you actually generate the file stubs with Rails (as opposed to just creating them by hand with a text editor).

Rails makes it easy to implement sweeping test database schema simply by specifying how to populate the database with a YML file. the `test/` directory has files that, according to a simple API, lets you run arbitrary requests over your Rails application, allowing comprehensive unit testing. I'm told that the stress testing is actually really simple also, but never have had to make one. The unit tests are actually trivial to run: `rake db:test:prepare`.

4.1.5 Statistics aggregation

We straightforwardly log all interesting user behavior in log files linked to the user. When a user clicks on something, we use the incredibly simple Rails logger interface to log it in the appropriate file. To retrieve the information, you can just pull the logs and process them locally. I personally use Hadoop for this purpose, even though it's really suited for much larger datasets.