REDUCING THE IMPACT OF CLIQUES IN SOCIAL NEWS SITES

ED KELLEY, DJ SHUSTER, BRIAN MATEJEK

1. Motivation

In 2004, Digg launched as one of the first popularity-based news sites on the Web. Instead of just a content delivery site, Digg was a platform where any user could submit a story and users voted to determine the order of the displayed stories. In the following years, similar sites, such as Reddit, StumbleUpon, and Hacker News, have seen a huge rise in popularity.

However, due to the nature of their voting systems, these sites are highly susceptible to collusion. There have been several instances of small groups of users, typically with strong political affiliations, colluding to "bury" or "downvote" content they disagree with and "digg" or "upvote" content that aligns with their politcal views. In 2007, Muhammad Saleem, a researcher for the Search Engine Journal, published an article titled "The Bury Brigade Exists, and Here's My Proof," showing that there were groups of users on Digg which were "hard at work burying any content that doesn't suit [their] ideology."[3,6]

Later, in 2010, AltNet published a story about the "Digg Patriots," a small group of conservative users that were "able to bury over 90% of the articles by certain users and websites submitted within 1-3 hours" and promote their own stories to the front page.[5] These users would gather on Yahoo Groups in order to collude on their voting.

This behavior was not limited to Digg. In 2012, The Daily Dot published a story about "LibertyBot," a bot that would use many accounts to downvote any posts seen as anti-libertarian or, more specifically, anti-Ron Paul.[4] These small groups of users are having a disproportionate impact on the content displayed on the front page of these social news sites.

2. Goal

Our goal is to find an algorithm that will reduce the impact of these cliques on the ranking of stories. This algorithm must be able to perform online with a large number of users and posts. Additionally, this algorithm should produce a ranking that places the most highly voted stories to the top, while diminishing the influence of biased cliques.

Date: 15 January 2013.

3. Reddit Ranking Algorithm

For comparison, we will be using the Reddit "Hot" Ranking algorithm. This algorithm assigns a a rating for each post which takes into account the difference between upvotes and down votes as well as the age of the post.

$s \leftarrow (ups - down)$
$order \leftarrow \log_{10} |s|$ ▷ By taking log, increase impact of early votes
if $order < 1$ **then**
    $order \leftarrow 1$
end if
if $s > 0$ **then**
    $sign \leftarrow 1$
else if $s < 0$ **then**
    $sign \leftarrow -1$
else
    $sign \leftarrow 0$
end if
$seconds \leftarrow (now - 1134028003)$ ▷ Time since an arbitrary date in 2005
return $order + \frac{sign * seconds}{45000}$ ▷ 45000 seconds = 12.5 hours

Figure 1. Pseudocode of Reddit's "Hot" Ranking Algorithm

4. Strategies

4.1. **Noisy Ranking.** The Gibbard-Satterthwaite Theorem expresses the idea that all deterministic voting rules are either dictatorial or non-strategy-proof. This means that either a single individual has the power to decide the entire election, or all agents have the incentive to lie about their true preferences. Neither of these scenarios is particularly attractive in a democratic selection process, a fact that has driven discussion toward the design of more favorable voting mechanisms.

One particular strategy, set forth originally by Gibbard and extended more recently by Conitzer and Sandholm, is the use of mechanisms that use randomized approximations.[2] These mechanisms are meant to mimic deterministic voting rules while adding a certain component of randomness. However, Gibbard also showed that these rules may only become strategy-proof in trivial scenarios. Nevertheless, the true advantage of this line of thinking is that it allows for the potential to develop voting mechanisms that may be considered approximately strategy-proof. Birrell and Pass first developed this term to accomodate the phenomenon that individuals tend

to report their preferences honestly when tactical voting provides only a small benefit. Birrell and Pass explain this phenomenon by postulating that there exists a psychological cost to lying. For this discussion, allow this value to be defined as epsilon. For a given value of epsilon, a voting rule is defined as being approximately strategy-proof if the benefit of misrepresentation is less than epsilon.[2]

Using these definitions as a framework, we sought to develop suitable randomized approximations of existing content-ranking algorithms. The first, and most simple, and of these voting rules involves adding noise to the voting process. Essentially, the algorithm introduces a certain probability of reversing each vote every time the posts score is reported. While this strategy may seem very simple, it provides an effective measure for combatting collusion. To begin, consider 3 types of users and posts (unbiased, bias1, bias2). Unbiased users upvote or downvote all posts using the same probability distributions, regardless of the posts biases. On the other hand, biased users will favor posts of their own bias, upvoting them with much higher probability than other posts. They are also much more likely to downvote posts that do not share their bias. Once noise is introduced, there is a certain probability that the users vote will counted in the opposite direction. Effectively, this reduces the overall benefit the user receives from casting his or her vote, as it is uncertain whether or not it will be cast favorably.

By following this line of thinking, noisy voting may also be a way to reduce the benefit received by mischaracterizing the users preferences. In our model, this reflects the undesirable effect of collusion. By adding noise, it is possible that the benefit of collusion may be brought below the epsilon value required for a voting rule to become approximately strategy proof . Another way to think of this effect is to see that the noise is effectively altering the users voting probability functions. In the simulations that we have run, unbiased users ignore a post with 60% probability, upvote it with 25% probability and downvote with 15% probability. Essentially, the expected score for any post is $.1 * n_{unbiased}$. Biased users, however, have only a 2% probability of ignoring a post with the same bias and a 2% probability of downvoting it, but a 96% probability of upvoting it. So, a biased post has an expected score of $.94 * n_{samebias}$. When viewing any other post, the biased users have a 40% probability of ignoring it, a 40% probability of downvoting it, and a 20% probability of upvoting it. This means that any post has an expected score of $-.2 * n_{differentbias}$. If there are two groups of biased individuals, each post will have the following expected scores:

$$E(unbiasedpost) = .1 * n_{unbiased} - .2 * n_{bias1} - .2 * n_{bias2} \tag{1}$$

$$E(bias1post) = .1 * n_{unbiased} + .94 * n_{bias1} - .2 * n_{bias2} \tag{2}$$

$$E(bias2post) = .1 * n_{unbiased} - .2 * n_{bias1} + .94 * n_{bias2} \tag{3}$$

$$Var(unbiasedpost) = .384 * n^2_{unbiased} + .544 * n^2_{bias1} + .544 * n^2_{bias2} \tag{4}$$

$$Var(bias1post) = .384 * n^2_{unbiased} + .079 * n^2_{bias1} + .544 * n^2_{bias2} \tag{5}$$

$$Var(bias2post) = .384 * n^2_{unbiased} + .544 * n^2_{bias1} + .079 * n^2_{bias2} \tag{6}$$

Clearly, biased posts will always have the upper hand in this scenario. However, noisy voting helps to combat this effect. Once noise is introduced, and there is a 20% probability for votes to be altered, unbiased users will still have a 60% probability of ignoring a post, but will instead have a 23% probability of having their vote recorded as an upvote and a 17% probability of having their vote recorded as a downvote. This makes each posts expected score equal to $.06 * n_{unbiased}$. Biased users will still have a 2% probability of ignoring a post of their own bias, but will instead have a 77.2% probability of having their vote recorded as an upvote and 20.8% chance of having it be recorded as a downvote. Now, a biased post will have an expected score of $.564 * n_{samebias}$. In addition, biased users will still have a 40% probability of ignoring a post with a different bias, but only a 36% probability of having their vote be recorded as a downvote and a 24% chance of having it recorded as an upvote. Now, any post will have an expected score of $-.12 * n_{differentbias}$. If there are two groups of biased individuals, each post will have the following expected scores:

$$E(unbiasedpost) = .06 * n_{unbiased} - .12 * n_{bias1} - .12 * n_{bias2} \tag{7}$$

$$E(bias1post) = .06 * n_{unbiased} + .564 * n_{bias1} - .12 * n_{bias2} \tag{8}$$

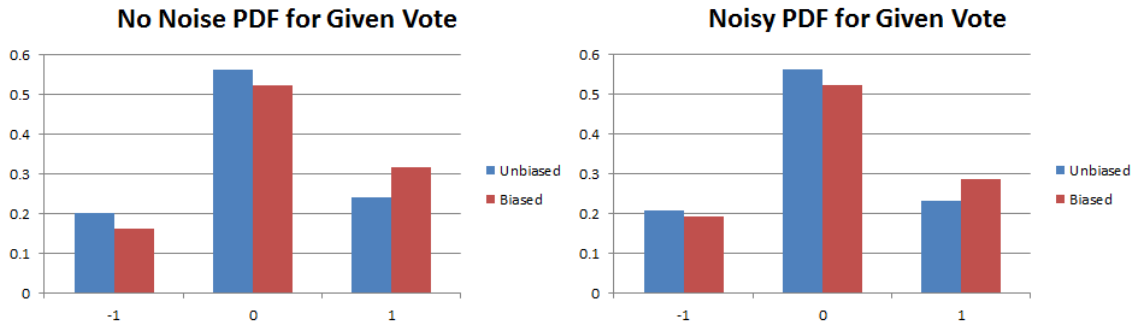
$$E(bias2post) = .06 * n_{unbiased} - .12 * n_{bias1} + .564 * n_{bias2} \tag{9}$$

$$Var(unbiasedpost) = .394 * n^2_{unbiased} + .580 * n^2_{bias1} + .580 * n^2_{bias2} \tag{10}$$

$$Var(bias1post) = .394 * n^2_{unbiased} + .645 * n^2_{bias1} + .580 * n^2_{bias2} \tag{11}$$

$$Var(bias2post) = .394 * n^2_{unbiased} + .580 * n^2_{bias1} + .645 * n^2_{bias2} \tag{12}$$

Once noise is introduced, all expected scores decrease, while variances of expected scores increase. While the expected scores decrease in equal proportions for both biased and unbiased posts, the variance of biased posts increases relatively greater than that of unbiased posts. However, since these probability distribution functions are not normal, we must plot them in order to achieve a full understanding of the results.



(A) No Noise (B) With Noise

Once the results are plotted, it appears that the added noise causes the biased and unbiased distributions to converge. Ultimately, this leads to a lesser likelihood for biased posts to appear on the front page than in the noseless case.

4.2. **Sampling.** Another method for collusion resistance is inspired by 'lottery voting'.[1] This voting mechanism works by choosing voters at random amongst a given population. These chosen few then cast ballots and determine the result of the election. While this sounds like a fairly radical idea, it is quite similar to the American jury system used to determine the guilt of an accused criminal on behalf of the people. In practice, lottery voting is believed to have several significant impacts on elections dynamics.[1] Since each individual vote has the potential to count for for a far greater share of the decision than in a traditional election, voters have the incentive to make more carefully reasoned decisions. In addition, lottery voting has a much more positive effect on maintaining proportional representation and cross-sectionalism in the voting group.[1] With this voting system, the choices of the minority actually matter and aren't simply dismissed by the majority as they are in our current "first past the pole" voting system. This also has the effect of increasing the viability of third party candidates that more accurately reflect a voter's preferences. In essence, lottery voting has the potential to be more truthful than traditional voting mechanisms.

As with the noisy case above, lottery voting is also a randomized approximation of a deterministic voting rule. Again, this allows us to explore the possibility for "approximate strategy-proofness". In our case, the algorithm works by first recording all upvotes and downvotes as normal. Then, each time the post's score is reported, the algorithm will randomly sample a portion of the post's votes. Then, this sample is extrapolated to represent the full volume of the post's votes. As with the lottery voting system, this sampling method allows for more accurate representation on the front page. Now, the percentage of unbiased posts on the front page should be more indicative of the total percentage of votes cast for unbiased posts. This should

reduce the positive benefit brought on by the collusive groups' tendency to cause biased posts to have slightly higher expected scores.

5. Simulations

We simulated the data over what would be an equivalent of 1 day, with 100, 300, and 1000 users. The simulations creates posts every minute and go throughs all of the users. The users can look at the last 3 hours worth of posts on the website (since users would rarely in real life go through many more posts than this). Users can either consciously not vote, up vote, or down vote a post. Once the user does this, the user will never vote again on the same post. Every thirty minutes the page rank algorithms of Reddit, Noisy, and Sampling are called which determine which pages make the front page. Front page is formally defined as the top 30 posts at any given time on our simulated social content website. The number of biased posts that make the front page is calculated for each of the simulations, and the following section describes proofs from this data.

We also ran simulations where there was only one collusive group. In 2010, the website Digg discovered a group of users called the Digg Patriots who gained a disproportionate amount of influence over the content of the website. We also looked at how our modified page rank algorithms handled this situation.

6. Analysis

We can form a binomial distribution of posts on the front page. The front page is calculated 47 times during the course of the simulation, and 30 posts are on each front page. If a post in a given slot on the front page is not biased, a random variable receives the value 1, otherwise it gets the value 0. Our null hypothesis (H_0) is that collusive groups or cliques do not have excessive influence on getting posts to the front page. Our alternative hypothesis (H_A) is that collusive groups do have excessive influence in getting posts to the front page. We will start by looking at the case where there are 100 users, 5 belong to one clique, 5 to another, and 85% of posts are not biased one way or the other. For H_0:

$$H_0 : \mu \geq 1198.5 \quad H_A : \mu < 1198.5$$
$$\text{n} = 1410 \quad \text{p} = .85$$
$$\mu = np = 1410(.85) = 1198.5 \quad \sigma^2 = np(1-p) = 1410(.85)(.15) = 179.8$$

After 50 trials with 100 users:

$$\mu_{reddit} = 766 \quad \mu_{sampling} = 1004 \quad \mu_{noisy} = 947$$
$$SE = \sqrt{\frac{\sigma^2}{n}} = \sqrt{\frac{179.8}{50}} = 1.896$$

$$Z_{reddit} = \frac{1198.5-766}{1.896} = 228.4 \quad Z_{sampling} = \frac{1198.5-1004}{1.896} = 102.6$$
$$Z_{noisy} = \frac{1198.5-947}{1.896} = 132.7$$

All three of these Z-indexes confirm that with 100 users, and 10 belonging to cliques, we can confidently replace the null hypothesis with the alternative hypothesis. Even with our algorithms, the probability of belonging to a clique is too great that the front page is still overly populated with their posts. However, noisy and sampling still do better than Reddit in terms of preventing posts from cliques from filling up the front page. For the following proofs, the null hypothesis (H_0) is that there is no difference in preventing cliques posts between the Reddit and sampling/noisy algorithms. The alternative hypothesis (H_A) is that sampling/noisy does better than Reddit. The means were determined for each of the 50 runs. We can create a μ_{diff} by subtracting means from the same run. First we will compare the Reddit and sampling algorithms.

$$H_0 : \mu_{reddit} = \mu_{sampling} \quad H_A : \mu_{reddit} < \mu_{sampling}$$
$$\mu_{diff} = 19.8 \quad \sigma_{diff} = 73.24 \quad SE_{diff} = 10.36$$
$$Z_{diff} = \frac{19.8-0}{10.36} = 1.91$$

We can say with over 97% confidence that the null hypothesis is wrong and that sampling works better than the reddit algorithm. Next, we will compare the reddit and noisy algorithms.

$$H_0 : \mu_{reddit} = \mu_{noisy} \quad H_A : \mu_{reddit} < \mu_{noisy}$$
$$\mu_{diff} = 15.05 \quad \sigma_{diff} = 55.81 \quad SE_{diff} = 7.89$$
$$Z_{diff} = \frac{15.05-0}{7.89} = 1.91$$

We can say with over 97% confidence that the null hypothesis is wrong and that sampling works better than the reddit algorithm. Lastly, we will compare the sampling and noisy algorithms, with the alternative hypothesis that the sampling algorithm is better than the noisy algorithm.

$$H_0 : \mu_{noisy} = \mu_{sampling} \quad H_A : \mu_{noisy} < \mu_{sampling}$$
$$\mu_{diff} = 4.75 \quad \sigma_{diff} = 18.37 \quad SE_{diff} = 2.60$$
$$Z_{diff} = \frac{4.75-0}{2.60} = 1.83$$

The p-value for this test is .0336, so we can say with over 96% confidence that sampling is better than the reddit algorithm. We also did two other tests (with 300 and 1000 users), and varied the number of users belonging to cliques. For the test with 300 users, we had 5% of users belonging to two different cliques (2.5% in each) and 4% of posts belonging to cliques. First, to see if colluding groups have disproportionate amount of influence (the alternative and null hypotheses are the same). The expected number of posts on the front page belonging to cliques is:

$$\text{n} = 1410 \quad \text{p} = .96$$
$$\mu = np = 1410(.96) = 1353.6 \quad \sigma^2 = np(1-p) = 1410(.96)(.04) = 54.14$$
$$H_0 : \mu \geq 1353.6 \quad H_A : \mu < 1353.6$$
$$\mu_{reddit} = 1236 \quad \mu_{sampling} = 1301 \quad \mu_{noisy} = 1286$$
$$SE = \sqrt{\frac{\sigma^2}{n}} = \sqrt{\frac{54.14}{50}} = 1.041$$
$$Z_{reddit} = \frac{1353.6-1236}{1.041} = 112.97 \quad Z_{sampling} = \frac{1353.6-1301}{1.041} = 50.53$$
$$Z_{noisy} = \frac{1353.6-1286}{1.041} = 64.94$$

These z-indexes are large enough that we can conclude that colluding groups still have a disproportionate amount of influence. However, again we want to see which (if any) of these algorithms is the best. We will start by comparing the sampling and Reddit algorithms. Our null hypothesis is that Reddit and sampling are comparable, and our alternative hypothesis is that sampling is better.

$$H_0 : \mu_{reddit} = \mu_{sampling} \quad H_A : \mu_{reddit} < \mu_{sampling}$$
$$\mu_{diff} = 5.39 \quad \sigma_{diff} = 20.1 \quad SE_{diff} = 2.84$$
$$Z_{diff} = \frac{5.39-0}{2.84} = 1.90$$

We can assert with over 97% confidence that our null hypothesis is wrong and the alternative, that sampling is better than Reddit's algorithm. Next, we will compare the Reddit and noisy ranking algorithms, with our null and alternative hypotheses the same as above.

$$H_0 : \mu_{reddit} = \mu_{noisy} \quad H_A : \mu_{reddit} < \mu_{noisy}$$
$$\mu_{diff} = 4.15 \quad \sigma_{diff} = 15.81 \quad SE_{diff} = 2.24$$
$$Z_{diff} = \frac{4.15-0}{2.24} = 1.85$$

This gives a p-value of .0322, so we can say with over 96% confidence that the noisy algorithm is better than the reddit algorithm with 300 users. Lastly, we will compare the sampling and noisy algorithms, with our null and alternative hypotheses the same as above.

$$H_0 : \mu_{noisy} = \mu_{sampling} \quad H_A : \mu_{noisy} < \mu_{sampling}$$
$$\mu_{diff} = 1.24 \quad \sigma_{diff} = 5.58 \quad SE_{diff} = .79$$
$$Z_{diff} = \frac{1.24-0}{.79} = 1.57$$

This gives a p-value of over .05. We cannot therefore conclude that the sampling algorithm is better than the noisy algorithm with 95% confidence. For 300 users, we do not have conclusive evidence if sampling or noisy is a better algorithm. We did an additional test with 1000 users, where the probability of any given post being biased is 96%. Again, we start by determining if collusive groups have a disproportionate amount of influence over the content that reaches the front page. First, to see if

colluding groups have disproportionate amount of influence (the alternative and null hypotheses are the same):

$$\text{n} = 1410 \quad \text{p} = .96$$
$$\mu = np = 1410(.96) = 1353.6 \quad \sigma^2 = np(1-p) = 1410(.96)(.04) = 54.14$$
$$H_0 : \mu \geq 1353.6 \quad H_A : \mu < 1353.6$$
$$\mu_{reddit} = 1140 \quad \mu_{sampling} = 1253 \quad \mu_{noisy} = 1228$$
$$SE = \sqrt{\frac{\sigma^2}{n}} = \sqrt{\frac{54.14}{50}} = 1.041$$
$$Z_{reddit} = \frac{1353.6-1140}{1.041} = 205.2 \quad Z_{sampling} = \frac{1353.6-1253}{1.041} = 96.64$$
$$Z_{noisy} = \frac{1353.6-1228}{1.041} = 120.65$$

These z-indexes also show that collusive groups still have a considerable influence on what content makes the front page. Next, as above, we will determine if either of these two new algorithms is more successful in keeping the front page free of biased content. We will start by comparing the sampling and Reddit algorithms. Our null hypothesis is that Reddit and sampling are comparable, and our alternative hypothesis is that sampling is better:

$$H_0 : \mu_{reddit} = \mu_{sampling} \quad H_A : \mu_{reddit} < \mu_{sampling}$$
$$\mu_{diff} = 9.39 \quad \sigma_{diff} = 33.25 \quad SE_{diff} = 4.70$$
$$Z_{diff} = \frac{9.39-0}{4.70} = 2.00$$

We can assert with over 97% confidence that our null hypothesis is wrong and the alternative, that sampling is better than Reddit's algorithm. Next, we will compare the Reddit and noisy ranking algorithms, with our null and alternative hypotheses the same as above.

$$H_0 : \mu_{reddit} = \mu_{noisy} \quad H_A : \mu_{reddit} < \mu_{noisy}$$
$$\mu_{diff} = 7.36 \quad \sigma_{diff} = 26.25 \quad SE_{diff} = 3.71$$
$$Z_{diff} = \frac{7.36-0}{3.71} = 1.98$$

We can say with over 97% confidence that the noisy algorithm is better than the reddit algorithm with 300 users. Lastly, we will compare the sampling and noisy algorithms, with our null and alternative hypotheses the same as above.

$$H_0 : \mu_{noisy} = \mu_{sampling} \quad H_A : \mu_{noisy} < \mu_{sampling}$$
$$\mu_{diff} = 2.02 \quad \sigma_{diff} = 7.99 \quad SE_{diff} = 1.13$$
$$Z_{diff} = \frac{2.02-0}{1.13} = 1.79$$

This gives a p-value just under .04. We can therefore justify saying that the sampling algorithm better eliminates the influence of cliques then the noisy algorithm.

7. Discussion and Conclusion

Social content sites are susceptible to collusive groups that try to manipulate which posts reach the front page. Even with the two methods that are collusion resistant, groups can still gain a disproportionate number of front page posts. The trouble arises from the fact the algorithms are run online - the interactions between users and how they will vote in the future are unknown. In our voting schemes, all the votes are tallied at once, but on a social content website, votes come in at different intervals. A page might make the front page shortly after being posted, with less than 10% of the total websites users seeing the content. Another issue with collusive groups is that their members tend to vote in force - on both posts that favor and oppose their opinions. Casual users of a social content website vote on fewer articles, which gives the collusive group even more power over the content of the front page. Our simulation takes this into consideration when determining how user's voted on content. In conclusion, Reddit's algorithm does not sufficiently offer protection against colluding groups or cliques of people. Users can gain a disproportionate amount of influence over the content of the front page, particularly the front pages of smaller subreddits. Both of the techniques we implemented were able to partially mitigate the influence of these groups.

8. Suggestions for Future Work

One of the aspects that we did not fully study was the *quality* of posts that reach the front page. That is, we carefully studied the distribution of biased vs. unbiased posts which reached the front page, but did not keep track of any "total front page score" based on the truthful views of unbiased users. By the criteria of biased vs. unbiased posts, an effective algorithm would be to just randomly chose the stories on the front page. However, this algorithm would not result in a high quality front page. Essentially, there is a tradeoff that should be studied between the attempts to reduce collusion and the desire to have a high quality front page.

9. Appendix

9.1. **main.py.** Creates users and posts, runs simulation, and prints results.

```
# simulates multiple days of posts and users in a social content website
import random
import sampling
import time
import reddit
import noisy

from user import User
from post import Post
from identify_bias import cluster_users

# number of days in the simulation
num_days = 1

# number of users
num_users = 100
num_posts = 4320

# possible groups a post or user can side with
no_group = 0
first_group = 1
second_group = 2

# probabilities of a post belonging to a collusive group
no_group_prob = .96
first_group_prob = .02
second_group_prob = .02

# probabilities of users belong to collusive groups
user_no_group_prob = .95
user_first_group_prob = .025
user_second_group_prob = .025

# textfile with all of the reports for each minute
textfile = file("SimulationReport.txt", "wt")

# run  the simulation
def main():
  for q in xrange(0, 1):
```

```
40    # an array of posts and users
41    content = [0] * num_posts
42    users = []
43    ranking = [0] * num_posts
44
45    # ids of posts and users
46    user_id = 0
47    content_id = 0
48
49    # initialize the number of users, create collusions
50    for i in xrange(num_users):
51      # determine user bias if any
52      user_bias = -1
53      user_bias_prob = random.uniform(0, 1)
54      if (user_bias_prob < user_no_group_prob):
55        user_bias = no_group
56      elif (user_bias_prob < user_no_group_prob + user_first_group_prob
            ):
57        user_bias = first_group
58      else:
59        user_bias = second_group
60      # create a new user
61      users.append(User(user_bias = user_bias, id = user_id, num_posts
            = num_posts))
62      user_id += 1
63
64    no_bias_sum = 0
65    first_bias_sum = 0
66    second_bias_sum = 0
67
68    no_bias_sum2 = 0
69    first_bias_sum2 = 0
70    second_bias_sum2 = 0
71
72    no_bias_sum3 = 0
73    first_bias_sum3 = 0
74    second_bias_sum3 = 0
75
76    # run simulation by the minute
77    iterate = 60 * 24 * num_days
78    blah = 0
79    for i in xrange(iterate):
80      # create three new posts every minute
```

```
81        number_new_posts = 3
82        # number of new posts a minute
83        for j in xrange(number_new_posts):
84          # determine post bias if any for new post
85          post_bias = -1
86          post_bias_prob = random.uniform(0, 1)
87          if (post_bias_prob < no_group_prob):
88            post_bias = no_group
89          elif (post_bias_prob < no_group_prob + first_group_prob):
90            post_bias = first_group
91          else:
92            post_bias = second_group
93          # create a new post
94          content[content_id] = (Post(post_bias = post_bias, id =
                content_id, time = i))
95          content_id += 1
96
97        # go through all users and see if they will upvote/downvote posts
98        for j in xrange(num_users):
99          var = random.uniform(0, 1)
100         if (var > 0.0):
101           # look at the past hour
102           last_hour = 60 * 3
103           if (content_id < last_hour):
104             for k in xrange(content_id):
105               users[j].vote(post = content[k])
106           else:
107             for k in xrange(last_hour):
108               w = content_id - k - 1
109               users[j].vote(post = content[w])
110
111       # call page rank algorithm for reddit
112       if (i % 30 == 0 and not i == 0):
113         posts_ranking = []
114         for j in xrange(content_id - 1):
115           post_data = [reddit.hot(content[j].ups, content[j].downs,
                content[j].date), content[j].id]
116           posts_ranking.append(post_data)
117
118         sorted_by_second = sorted(posts_ranking, key=lambda tup: tup
              [0], reverse = True)
119
120         no_bias_number = 0
```

```
121         first_bias_number = 0
122         second_bias_number = 0
123
124         for b in xrange(0, 30):
125           id = sorted_by_second[b][1]
126           if (content[id].post_bias == no_group):
127             no_bias_number += 1
128           elif (content[id].post_bias == first_group):
129             first_bias_number += 1
130           else:
131             second_bias_number += 1
132         no_bias_sum += no_bias_number
133         first_bias_sum += first_bias_number
134         second_bias_sum += second_bias_number
135
136         posts_ranking = []
137         for j in xrange(content_id - 1):
138           post_data = [sampling.hot(content[j].ups, content[j].downs,
                content[j].date), content[j].id]
139           posts_ranking.append(post_data)
140
141         sorted_by_second = sorted(posts_ranking, key=lambda tup: tup
              [0], reverse = True)
142
143         no_bias_number = 0
144         first_bias_number = 0
145         second_bias_number = 0
146
147         for b in xrange(0, 30):
148           id = sorted_by_second[b][1]
149           if (content[id].post_bias == no_group):
150             no_bias_number += 1
151           elif (content[id].post_bias == first_group):
152             first_bias_number += 1
153           else:
154             second_bias_number += 1
155         no_bias_sum2 += no_bias_number
156         first_bias_sum2 += first_bias_number
157         second_bias_sum2 += second_bias_number
158
159         posts_ranking = []
160         for j in xrange(content_id - 1):
```

```
161             post_data = [noisy.hot(content[j].ups, content[j].downs,
                    content[j].date), content[j].id]
162             posts_ranking.append(post_data)
163
164         sorted_by_second = sorted(posts_ranking, key=lambda tup: tup
                [0], reverse = True)
165
166         no_bias_number = 0
167         first_bias_number = 0
168         second_bias_number = 0
169
170         for b in xrange(0, 30):
171           id = sorted_by_second[b][1]
172           if (content[id].post_bias == no_group):
173             no_bias_number += 1
174           elif (content[id].post_bias == first_group):
175             first_bias_number += 1
176           else:
177             second_bias_number += 1
178         no_bias_sum3 += no_bias_number
179         first_bias_sum3 += first_bias_number
180         second_bias_sum3 += second_bias_number
181
182     total = no_bias_sum + first_bias_sum + second_bias_sum
183     print '-----Reddit-----'
184     print float(no_bias_sum) / total
185     print float(first_bias_sum) / total
186     print float(second_bias_sum) / total
187     print '----Sampling----'
188     print float(no_bias_sum2) / total
189     print float(first_bias_sum2) / total
190     print float(second_bias_sum2) / total
191     print '-----Noisy------'
192     print float(no_bias_sum3) / total
193     print float(first_bias_sum3) / total
194     print float(second_bias_sum3) / total
195     print no_bias_sum + first_bias_sum + second_bias_sum
196 if __name__ == '__main__':
197   start_time = time.time()
198   main()
199   print time.time() - start_time
```

9.2. **user.py.** User class. Contains voting logic.

```
# User class for simulation
import random

# possible votes
start = -2
downvote = -1
nothing = 0
upvote = 1

# possible groups a post or user can side with
no_group = 0
first_group = 1
second_group = 2

class User(object):
  def __init__(self, user_bias, id, num_posts):
    self.id = id                          # unique user id
    self.user_bias = user_bias            # is the user in a collusion
    self.est_bias = 0
    self.voting_history = [start] * num_posts  # initialize a voting
        history array

  def vote(self, post):
    # if the user has not voted on this post
    if (self.voting_history[post.id] == -2):
      if (self.user_bias == 0):
        var = random.uniform(0, 1)
        # ignore 60% of the time
        if (var < 0.60):
          self.voting_history[post.id] = nothing
        # upvote 25% of the time
        elif (var < 0.85):
          self.voting_history[post.id] = upvote
          post.upvote()
        # downvote 15% of the time
        else:
          self.voting_history[post.id] = downvote
          post.downvote()
      else:
        # if biases are the same, with 99% probability upvote
```

```
40          if (post.post_bias == self.user_bias and random.uniform(0, 1) <
                0.95):
41            self.voting_history[post.id] = upvote
42            post.upvote()
43          # else downvote with probability 40%, ignore 40%, upvote 20%
44          else:
45            var = random.uniform(0, 1)
46            if (var < 0.40):
47              self.voting_history[post.id] = downvote
48              post.downvote()
49            elif (var < 0.80):
50              self.voting_history[post.id] = nothing
51            else:
52              self.voting_history[post.id] = upvote
53              post.upvote()
```

9.3. **post.py.** Post class.

```
# class of posts for a social content website.  Posts can have bias.
import random
import datetime

class Post(object):
  def __init__(self, post_bias, id, time):
    self.post_bias = post_bias  # bias of the post
    self.id = id                # id of the post
    self.ups = 0          # number of upvotes
    self.downs = 0          # number of downvotes

    hour = time / 60
    min = time % 60

    day = hour / 24 + 13
    hour = hour % 24

    date = datetime.datetime(2013, 1, day, hour, min)
    self.date = date       # date of the post

  def upvote(self):
    self.ups += 1

  def downvote(self):
    self.downs += 1
```

9.4. **reddit.py.** Actual Reddit ranking algorithm.

```
#Rewritten code from /r2/r2/lib/db/_sorts.pyx

from datetime import datetime, timedelta
from math import log

epoch = datetime(1970, 1, 1)

def epoch_seconds(date):
  #"""Returns the number of seconds from the epoch to date."""
  td = date - epoch
  return td.days * 86400 + td.seconds + (float(td.microseconds) /
      1000000)

def score(ups, downs):
  return ups - downs

def hot(ups, downs, date):
  #"""The hot formula. Should match the equivalent function in postgres
      ."""
  s = score(ups, downs)
  order = log(max(abs(s), 1), 10)
  sign = 1 if s > 0 else -1 if s < 0 else 0
  seconds = epoch_seconds(date) - 1134028003
  return round(order + sign * seconds / 45000, 7)
```

9.5. **noisy.py.** Noisy algorithm.

```
#reddit algorithm + noise

from datetime import datetime, timedelta
from math import log
import random

epoch = datetime(1970, 1, 1)
noise = .20

def epoch_seconds(date):
    #"""Returns the number of seconds from the epoch to date."""
    td = date - epoch
    return td.days * 86400 + td.seconds + (float(td.microseconds) /
        1000000)

def score(ups, downs):
    new_ups = 0
    new_downs = 0
    for x in range(0, ups):
            noise_prob = random.uniform(0,1)
            if (noise_prob < noise):
                    new_downs += 1
            else:
                    new_ups += 1

    for y in range(0, downs):
            noise_prob = random.uniform(0,1)
            if (noise_prob < noise):
                    new_ups += 1
            else:
                    new_downs += 1

    return new_ups - new_downs

def hot(ups, downs, date):
    #"""The hot formula. Should match the equivalent function in
        postgres."""
    s = score(ups, downs)
    order = log(max(abs(s), 1), 10)
    sign = 1 if s > 0 else -1 if s < 0 else 0
    seconds = epoch_seconds(date) - 1134028003
```

```
40      return round(order + sign * seconds / 45000, 7)
```

9.6. **sampling.py.** Sampling algorithm.

```
#reddit algorithm + sampling / extrapolation

from datetime import datetime, timedelta
from math import log
import random

epoch = datetime(1970, 1, 1)

def epoch_seconds(date):
  #"""Returns the number of seconds from the epoch to date."""
  td = date - epoch
  return td.days * 86400 + td.seconds + (float(td.microseconds) /
      1000000)

def score(ups, downs):
  votes = []
  n = (ups + downs) / 4
  for x in range(0,ups):
      votes.append(1)
  for y in range(0,downs):
      votes.append(-1)

  sample_votes = random.sample(votes, n)

  return (sum(sample_votes) * 4)

def hot(ups, downs, date):
  #"""The hot formula. Should match the equivalent function in postgres
      ."""
  s = score(ups, downs)
  order = log(max(abs(s), 1), 10)
  sign = 1 if s > 0 else -1 if s < 0 else 0
  seconds = epoch_seconds(date) - 1134028003
  return round(order + sign * seconds / 45000, 7)
```

References

[1] Akhil Reed Amar. Choosing representatives by lottery voting. *Yale Law Journal*, 93, 1984.

[2] Eleanor Birrell and Rafael Pass. Approximately strategy-proof voting. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 67–72. AAAI Press, 2011.

[3] David Cohn. Hunting down digg's bury brigade. `http://www.wired.com/techbiz/people/news/2007/03/72835`, March 2007.

[4] Kevin Morris. How bots silence ron paul critics and threaten the democracy of reddit. `http://www.dailydot.com/society/ron-paul-liberty-downvote-bot-reddit/`, May 2012.

[5] Ole Ole Olson. Massive censorship of digg uncovered. `http://blogs.alternet.org/oleoleolson/2010/08/05/massive-censorship-of-digg-uncovered/`, August 2010.

[6] Muhammad Saleem. The bury brigade exists, and here's my proof. `http://www.dailydot.com/society/ron-paul-liberty-downvote-bot-reddit/`, February 2007.

[7] Amir Salihefendic. How reddit ranking algorithms work. `http://amix.dk/blog/post/19588`, November 2010.

This paper represents our own work in accordance with university regulations.