THE BLIND SPAM-MAKER

Machine Learning Term Project

Elizabeth Gifford
CS 113 - Spring 2007

Goals

The main goal of this experiment is to set up a structure for evolving a complicated fake e-mail that would be able to bypass a trained bayesian e-mail filter. The premise is that while humans might leave gaping holes in traditional spam filters that spammers can later exploit, it might be beneficial to look for these holes in a systematic and non-human way so that some weaknesses might be detected before the filter is released, and thus before the spammers would be able to find them. It is not the purpose of this paper to come up with new rogue ways to bypass spam filters for the benefit of spam as a whole, but rather to demonstrate that a learning technique such as the natural selection evolution presented in The Blind Watchmaker can find areas that spammers might find to exploit that filter designers may not have anticipated. This could be compared to the practice of hiring an "ethical hacker" to try to break into your secure system to test how secure it really is, but in this particular case, the hacker isn't a human.

Background

E-Mail Spam

Anyone who has an e-mail address these days has experienced e-mail spam. E-mail spam is defined on the Wikipedia entry as, "a subset of spam that involves sending nearly identical messages to numerous recipients by e-mail. Spam is e-mail that is both unsolicited by the recipient and sent in substantively identical form to many recipients." Spam is frustrating to many users because as the amount of spam in their inboxes grows, it becomes more difficult to cull out important e-mail. As e-mail has increased in popularity, so has spam. According to the same Wikipedia article, the first large-scale spam was sent to 6000 newsgroups in 1994, and by February 2007, 90 billion spam messages are being sent per day.

To deal with the ever-increasing amount of spam e-mail, there has been also a proliferation of software designed to automate the process of spam e-mail detection and filtering. Many popular e-mail programs now include a filter that can be turned on and off, and made stricter or less strict through user preferences. Some filters can learn about a user's e-mail

preferences over time as the user marks e-mails as either "spam" that they wish to discard or "ham" that they wish to read. Certain attributes of these e-mails are analyzed by the filtering algorithm, and it eventually learns to tell what the user considers to be good and bad e-mail.

Many spam filtering programs use a naive bayesian classifier to identify spam (Wikipedia: Bayesian Spam Filters). These are useful because they can be trained by individuals to recognize e-mails that the user is likely to receive. Probabilities are calculated for a large corpus of words by "training" the filter on a large variety of e-mails that have already been identified as either "spam" or "ham." A word frequently seen in spam, such as "money" would have a higher probability than words you might see in real e-mails from your friends, like their names. Once the filter has been trained, it can perform a statistical analysis on incoming e-mails, and assign each e-mail a probability that represents the likelihood that the e-mail in question is legitimate. Once a spam filter like this has been activated, it can automatically move any suspicious e-mails to another folder for users to review so that spam e-mails do not clutter up the inbox.

Spammer tactics

Since e-mail filters need to be public knowledge for users to know about and use them, it is inevitable that spammers know about them as well. As e-mail filters become more complex and better at filtering e-mails, spammers crack the filters and find new ways to bypass them. In his lecture at ToorCon 2004, Justin Mason of SpamAssasin talked about the iterative design of spam. Over the years, spam has gone from easily discovered blatant e-mails to impersonating legitimate e-mail, to adding random gibberish or quotes of legitimate english to the e-mail bodies. Bayesian filters can be updated to detect new spam as it evolves, but this has turned into an arms race.

Spam filters get better, and so does spam. However, it can be difficult to figure out what tactics spam is using because the software and algorithms used by spammers are not public. As a result, spam filter engineers are left trying to backwards engineer the spam making process, and frequently end up one step behind the spammers. The filters are written, there are inevitably holes in the filters that people hadn't thought of, spammers find the holes and exploit them, and filter writers are left trying to plug the holes later.

An interesting part of this is that holes in filters can occur due to the oversight of people. However, what if we had a way for something other than a *person* to look for these holes? Instead of trying to backwards engineer the spammer techniques, we could instead try to backwards engineer the *filters themselves* to look for flaws they might have. An important as-

pect of this technique might be to take the human perspective on e-mail out of the loop as much as possible, and let machines look for flaws in the filters.

The Blind Watchmaker and Finding Holes

In his 1986 book, The Blind Watchmaker, Richard Dawkins presents an argument for natural selection. In his descriptions, Dawkins argues that the high complexity of natural biology is not due to a divine hand, but rather a collection of coincidences. He calls natural selection a "blind watchmaker" because the highly complex process does not plan ahead during development, but rather components of biological specimens change randomly. If they change for the good, they survive in later generations, but if the changes are not useful to a species, the mutated specimens simply die out. This gives the impression that current organisms might have been designed intelligently, but what really happens is that all of the mistakes are simply flushed out of existence.

This basic principle could also be applied to spamming techniques. Instead of having an intelligent designer, or human controller, try to decide what might constitute spam and what a filter might be able to detect, a complex spam corpus can be evolved to survive in the "environment" of a spam filter. By looking at the "survivors" in this sort of environment, spam filter designers could detect flaws or poor assumptions in the filter design and improve on the filters before spammers find the same flaws to exploit. This would still be a necessarily iterative process, but it might make filters more difficult for spammers to crack in the future.

Design

The basic actions for this experiment involve building up a set of generic formuli that can be used to write messages with similar characteristics, writing multiple messages with each formula so that the average classification from the filter can be calculated, sending all of the messages through a filter that is treated like a black box algorithm so that the human coder can't design specifically for the spam filter, calculating the results, saving the "best" formuli and spawning variants of those formuli, and starting the loop over. There are several components to the design of this experiment. The evolution coordinator controls all of the following: the email formula builder, the spam writer, the spam filter, the spam tester, and the formula evolver. Each component works as follows, and the entire set of components is iterated through for each generation of the spam building cycle.

The Formula Builder

The formula builder puts together a set of formulas that can be used to generate emails at a later time. These could be seen as the genome of the spam, carrying encoding information about how the message will behave as it is constructed. Each spam formula is a string consisting of a random sampling from the set [word, advertise, capitals, scramble, space, link, punctuate]. When translated into messages, each of these tokens gives a specific direction for how the message should be written.

The Spam Writer

The spam writer reads in a formula constructed by the formula builder, and translates it according to the following rules. The tokens in the string are stepped through one by one, and a new string containing the spam message is created. The *word* token is replaced by a random English word from a small hand-selected dictionary, chosen so that it does not include any words that sound like advertisements. The *advertise* token is replaced by a random English word from a separate dictionary, hand-populated solely with words that are typically found in advertisements. The *capitals* token invokes an algorithm that steps through the parts of the message that have been created thus far and replaces 30% of the letters with a capital letter. The *scramble* token is the only trick employed by the genome itself that isn't evolved. It plays off of a popular internet meme, based on the unpublished thesis work of Graham Rawlinson, that proposes that humans can still read words with transposed letters as long as the first and last letters remain in place. This could be important to spam because a scrambled word that is recognizable to a human might fool a machine. As such, this token causes the word directly prior to it to be scrambled save for the first and last letter. The *space* token appends a space to the string, the *link* token appends a url generated from the natural English dictionary mentioned above, and the *punctuate* token appends a punctuation symbol to the string.

The Spam Filter

I also needed a spam filter to test on. However, I didn't want to write this myself because I wanted to avoid any bias that might creep into my code if I knew how the filter worked. I decided to look for an open source spam filter that I could easily interface with, and use it in my experiment without reading about how it worked first so that my aspects of the blind spam maker could be as clear of bias as possible. Looking through SourceForge, I found an open source python package called SpamBayes. I read only enough of the FAQs to figure out how to install it and how to train the filter and pass messages to it to be scored before running my experiments. After I had finished with the experimentation, I read more about how this particular filter works so that I could summarize it here. In training the filter, I fed

it a spam corpus used in a previous assignment for an AI class. The corpus contained several hundred spam emails that had been pre-tagged as spam, and several hundred emails from various programming discussion lists. After being trained, the filter can be fed messages and will return a number between 0 and 1 that signifies its classification of the email. A zero would mean that the filter was completely sure that the email was spam, a 1 means that it is entirely sure the email is ham, and ratings hovering around 0.5 signify that the filter has no idea whether the email is spam or ham.

SpamBayes is a relatively standard bayesian network spam classifier. It is originally based off of Paul Graham's essay, "A Plan for Spam." He recommends that the best strategy for stopping spam is through content-based filters. These work, as described above, by assigning weights to various words in an email based on how likely each word is to show up in a spam or ham email. The email is then classified based on its total weights. He notes that this statistical technique is far better than the archaic method of writing rules based on the basic characteristics of spam (punctuation, capitalization, etc). You will note that this is fairly contrary to the techniques I chose to base my spam evolution on. This is due partly to my desire to treat the spam filter as a black box that I knew nothing about so that the algorithm itself could find holes; partly to the need to simplify my own algorithm for a first attempt since I didn't want to try to emulate natural language or even more complicated structures on a first pass; and partly to my own novice view of the world of spam: at first, when you haven't examined the workings of modern spam filters, the rule-based classification seems like a great idea!

The Spam Tester

The spam tester in my experiment took each formula, created some number of spam messages (between 20 and 100, depending on the experiment I was running) from it using the spam writer described above, and then took an average of all of the ratings given to emails generated by a given formula. The tester then sorts all of the formulas by this value and keeps a fixed percentage of the "best" filters while discarding the rest.

The Spam Evolver

The final step in this process is the spam evolver. Once a set number of formuli have been discarded, the remaining formuli are fed into the spam evolver. Each formula in the set is paired up with two other formuli and "mated" according to a set of weights that I tweaked by hand over the course of the experiments. A child token string is formed where for each token in one of these parent formuli, a token can be deleted, inserted randomly, repeated, or copied from either parent. This is similar to how the human genome is replicated.

Experimentation

Part 1: Sample Size

Once all of these features of the spam evolution system had been constructed and fitted together, I performed several experiments to see how well this system could work.

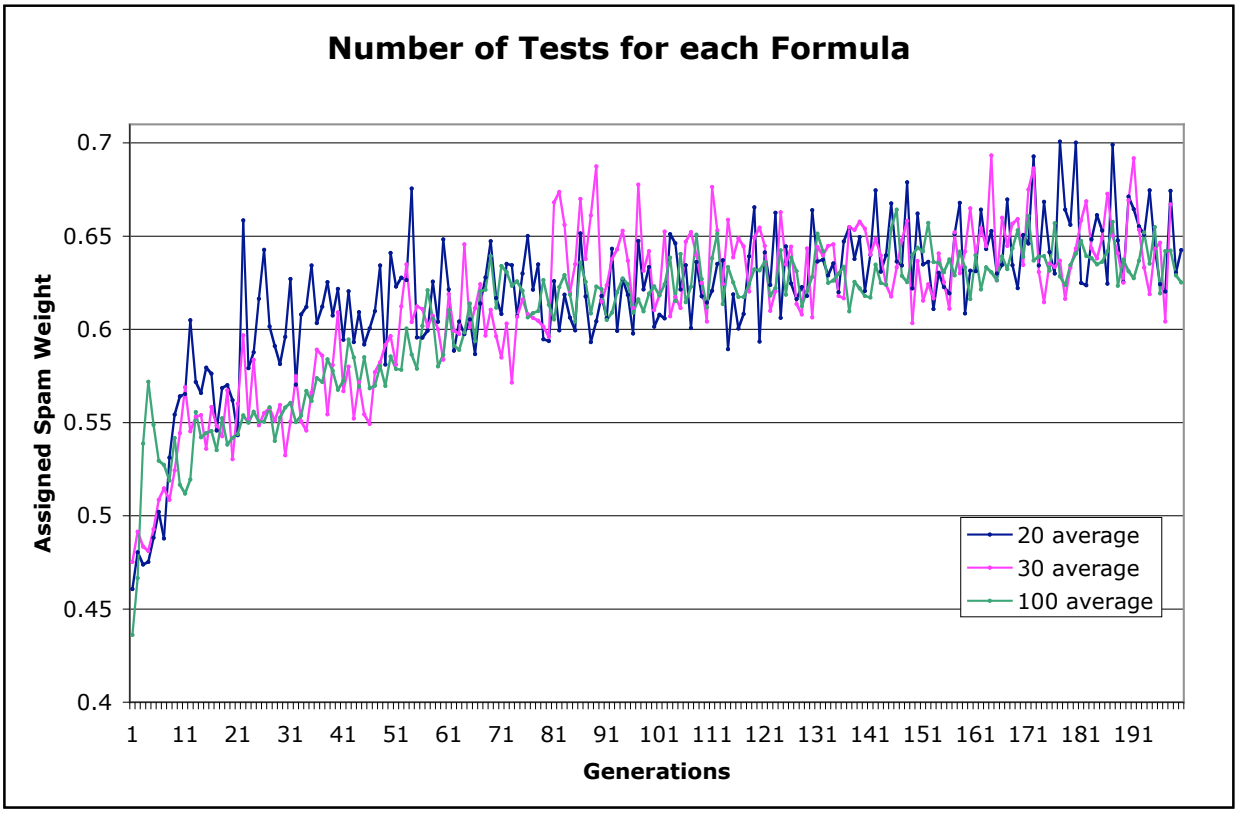

Figure 1: Note that the Y axis has been truncated to get a better view of this distribution. While none of the different test sizes performed drastically better, the 100 average trials produce a smoother line.

In the first try, I assigned weights to each of the options in the formula evolver at will. Fifty formuli were randomly constructed, tested, culled, and then the actual evolution process started. Each time a formula was judged, 30 messages spawned by that formula were judged in the spam filter and averaged. The results from this trial can be seen in Figure 1, where the best scoring formula's score from each iteration is plotted in blue. As described in the spam filter section, a score of 0 would be a score that is certainly spam while a score of 1 would indicate that the filter believed the message was certainly ham.

In an attempt to smooth this line out and gain better results, I decided to average over more trials of each formula. I repeated the same experiment, averaging instead over 30 and then

100 trials of each formula. These trials are also shown in Figure 1. For all three sample sizes, the formuli were evolved over 500 generations.

In part 1 of the experiment, the mating part of the evolution algorithm is as follows: 20% of the time a token is passed over, it is deleted. 32.5% of the time, the token is copied from *either* parent 1 or parent 2. 22.5% of the time, a random token is inserted instead of either parent. 23% of the time, the tokens from both parents are appended, and 2% of the time a token from one parent or the other is repeated a random number of times.

Part 2: More Vocabulary

The experiment started with just a handful of English vocabulary. Thinking that such a limited word list might be harming the overall performance of the algorithm, the next test I tried was to add vocabulary to the dictionary. I added about 200 words, randomly selected using a random word generator written by "Howard" at "Howard's Musings." All other factors of the experiment were kept the same. It appears that from running several trials with an expanded vocabulary, this did not offer marked improvement to the algorithm.

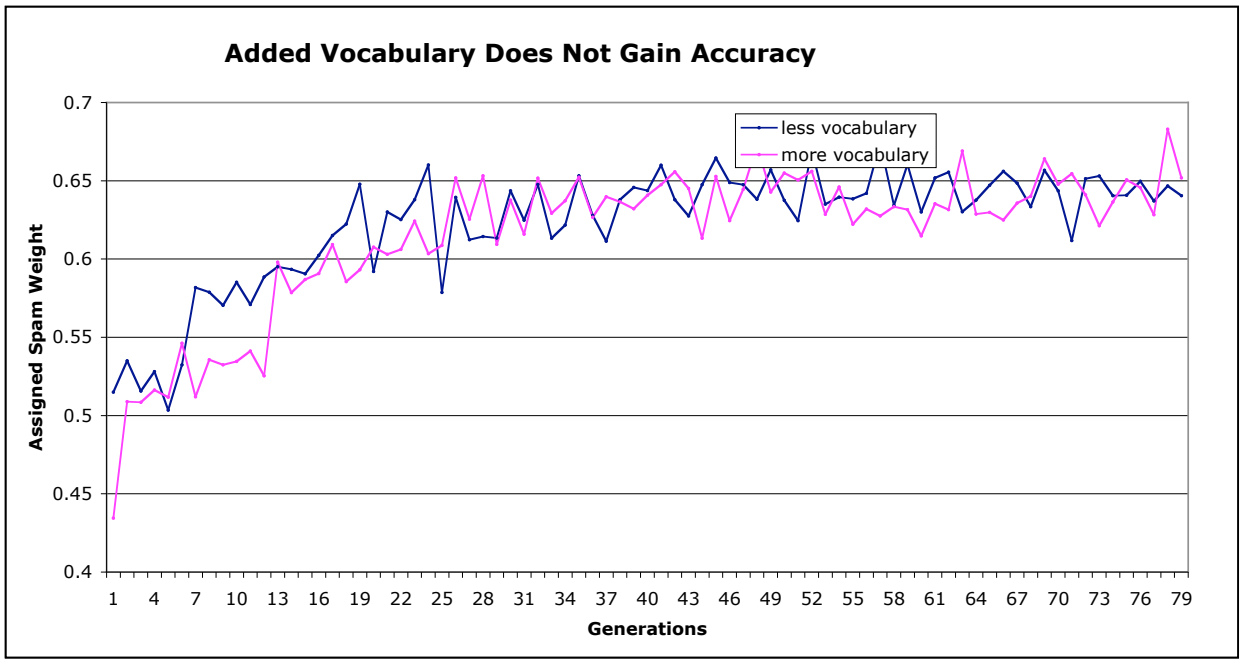

Figure 2: Added vocabulary does not affect the overall performance. Note, again, that the Y axis has been truncated to show the performance difference better.

Part 3: Longer Trials

Next, I ran several trials with a duration longer than 500 generations to see whether running the evolution for a longer period of time resulted in spam that could more easily bypass the filter. As seen in Figure 3, this did not substantially increase the likelihood of generated messaged to be classified as ham, so I continued the experiment with relatively short generation lengths so that more different trials could be run in a limited amount of time rather than fewer longer runs.

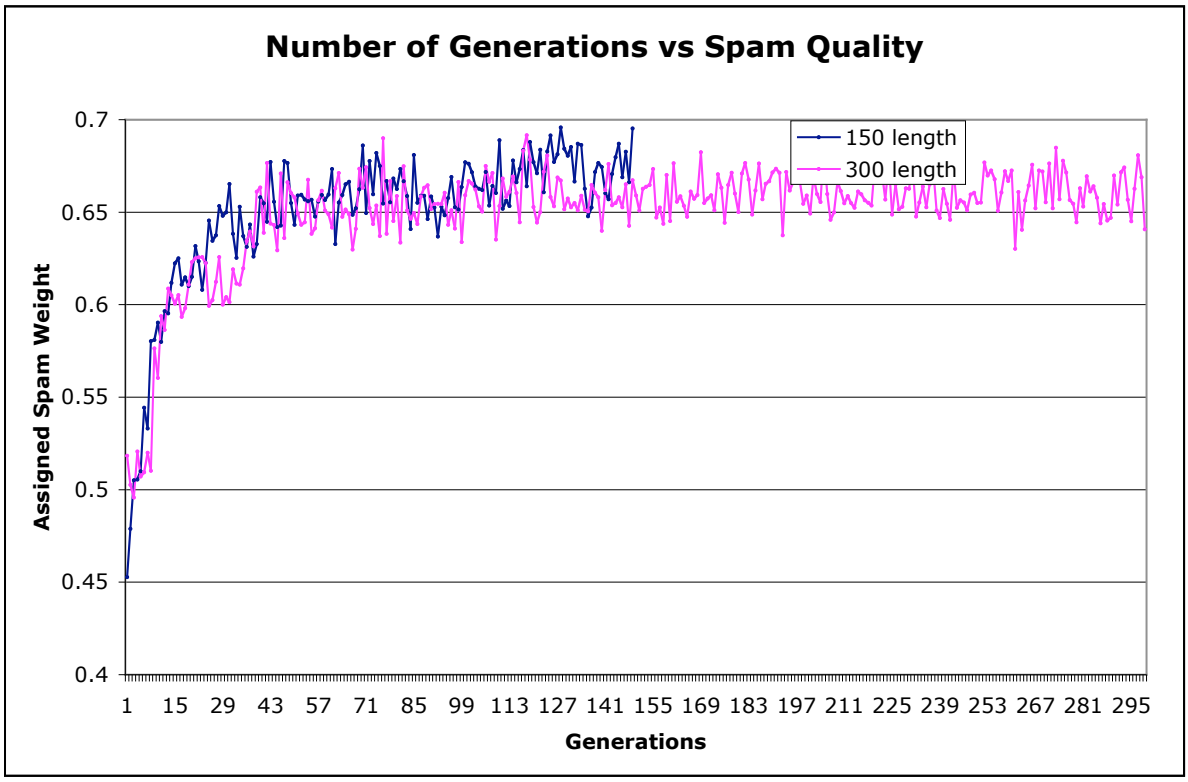

Figure 3: This graph shows that doubling the number of generations does not gain much in the way of ham ratings. Once again, the Y axis has been truncated.

Part 4: Varying Weights

For this segment, I varied the weights of the evolution section over several different combinations. For example, I noticed that the spam had a tendency to grow iteratively longer because there was a much heavier weight placed on actions that would make the strings grow longer than on actions that would keep them the same length or shrink them. First I sought to even these actions out, and then I varied the weights over several other different combinations of values. Table 1 shows the various combinations of weights, and Figure 4 shows the results from each of those combinations. Clearly, the changes in weights has an effect on the performance of each algorithm overall.

	1st trial	2nd trial	3rd trial	4th trial
delete token	20	22	16	14
append parent 1 or 2	32.5	38	44	46
append random token	22.5	22	16	14
append both parents	23	16	22	24
iterate 1 parent n times	2	2	2	2

Table 1: This table shows the probability that any of the above 5 actions was taken on each token in a pair of spam formuli. For example, when "mating" two formuli under the 1st trial set of weights, 20% of the time the token was deleted.

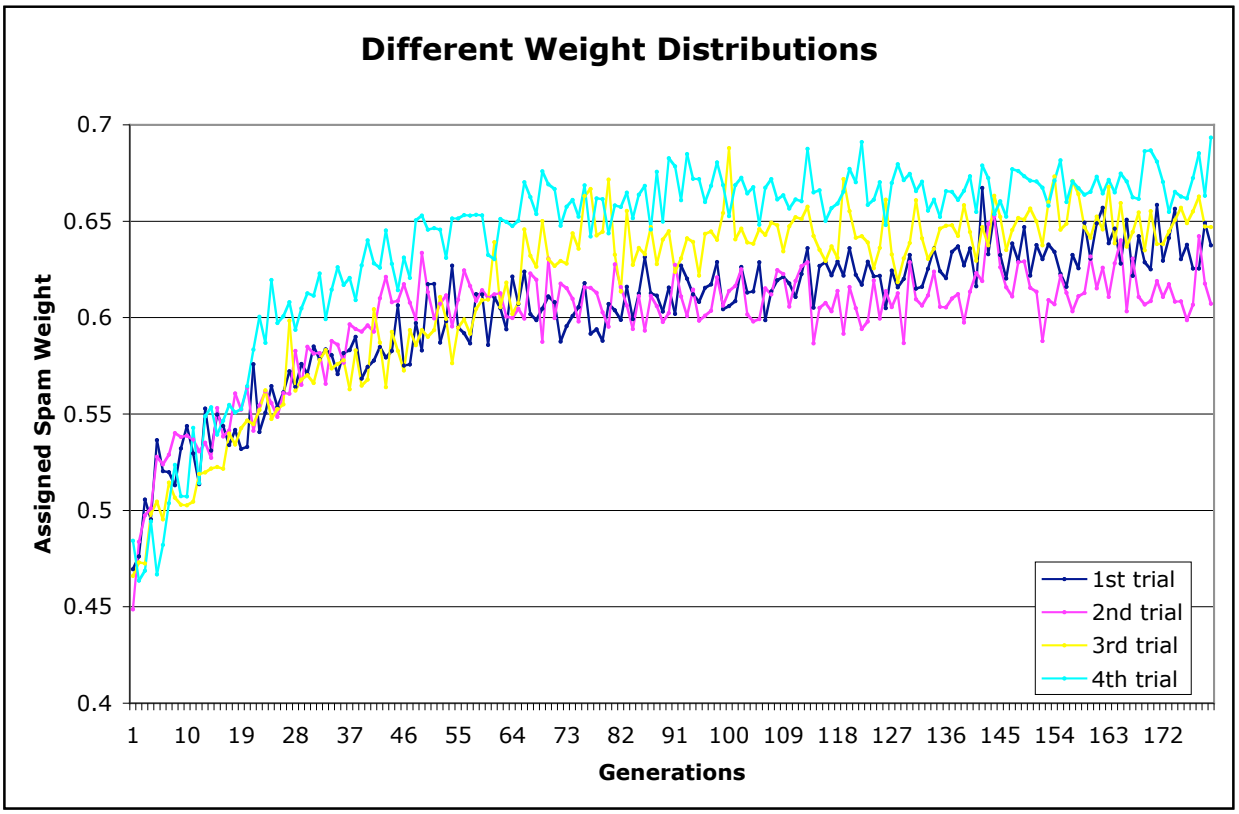

Figure 4: This figure shows the spam/ham weight of the "best" message formula in each generation for the four different weight distributions described above. Note that the Y axis has been truncated to better show the distribution.

Evolving Spam

Finally, the plot in Figure 5 is a demonstration of how much easier it is to evolve something that might get caught in a spam filter than something that might not get caught in a spam filter. Rather than saving the *best* formuli of each generation, this time I selected the *worst* formuli each time. This curve levels off as well, but it is much closer to 0 than the best evolved formuli ever were to 1. The curve is also much steeper.

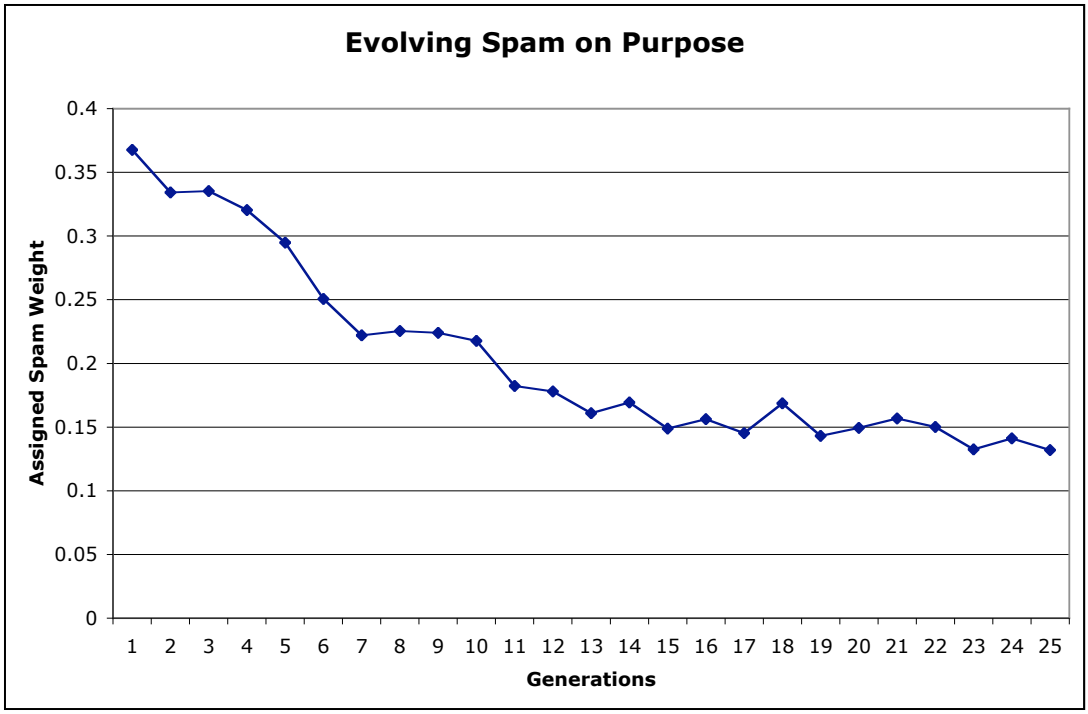

Figure 5: This plot demonstrates that actual spam emails can be evolved much more quickly, should we want to actually create something that would be caught by a filter.

Resulting Spam

Below are examples of two different messages evolved using the techniques detailed in this paper. It turns out that in the limited set of tests performed for this paper, the highest performing messages could be created by averaging over 100 trials for each spam, making the dictionary for words larger, and balancing the evolution weights so that strings grew, but grew slowly instead of at a very rapid pace. The messages shown below look strange to the casual human observer, but the main exercise in this paper was to evolve a message that was certainly spam, but would be able to bypass the filter in place to cause the disruption and annoyance that spam is intended. However, even with that in mind, the messages produced by the higher scoring formuli are still much more sensible to a human reader than the messages produced by lower scoring formuli.

This message was rated at 0.35, or more likely to be rated as SPAM than HAM.

```
ELNHBULTC   TATT.KDRATOT:/HMIEIHHBYU
TWW.CIRAOTWEWTIIHSRHTTP:ATEEONIHSCOMYT
```

This message was rated at 0.68, or more likely to be rated as HAM than SPAM:

```
?   http://www.willowdamsel.com   low   .  buy
http://www.tampersbruiting.com   mnemonic   gleannewflagellum
http://www.tickledundamaged.com stock http://www.monumentspiccolos.com
new  splashily  http://www.briefcasedeftly.com
http://www.foldingbackstops.com % enlarge
http://www.gloverequester.com ?         free legal . detonatesnew
http://www.ballastwhistlers.com rate  http://www.iodizespiel.com !
http://www.coltsownership.com refinance!  http://www.thefolding.com
detail http://www.foaminessimplored.com legalpayment
http://www.southpawinitialed.com stock refinance
http://www.bruitingbijoux.com like? http://www.felldetonates.com
http://www.boughhenna.com legal  low  http://www.abashedcreakily.com
http://www.eventfulmuzzled.com  \m
```

Conclusion

Overall, a perfect spam was not found using this technique. However, evolution times were short and the variety of genomes used here was very limited. If this program were scaled up to include more detailed aspects of emails (like headers, signatures, more content, etc), and more detailed morphing genes like the scramble token, I believe that higher overall scores could be reached. A similar technique could also be performed using just a large dictionary that could bypass filters that work more like the one used here that examines content rather than structure. However, one drawback of that technique would be that extremely specific emails would be generated, and the blockers for them would have to be equally specific.

I think that an expansion of the technique above would be better able to showcase the shortcomings of particular spam filters. For example, the simplistic technique demonstrated in this paper shows that f this particular spam filter (that has been trained on relatively "old" spam) can be exploited by giving it random gibberish as has become a favored technique of spammers more recently than this spam corpus was developed. It is not conclusive whether this technique would only be useful for testing the ability of filters to stand up to gibberish, or whether more complicated weaknesses could be found with a more complex and longer running evolutionary system. Despite the failure of most of my changes to make much of a different in performance, all of the versions did improve performance over the generations as they ran.

BIBLIOGRAPHY

1. Dawkins, Richard. The Blind Watchmaker. Longman: England 1986.
2. Graham, Paul. "A Plan for Spam" published in Hackers and Painters: Big Ideas from the Computer Age. O'Reilly Media, Inc. May 2004.
3. Hansen, Howard. Random English Word Picker, Howard's Musings. http://www.howardsmusings.com/randomwords.html
4. Mason, Justin. Spam Forensics: Reverse-Engineering Spammer Tactics, Toorcon 2004: HTML slideshow from MagicPoint. <http://spamassassin.apache.org/presentations/2004-09-Toorcon/html/>
5. Rawlinson, G. E. (1976) The significance of letter position in word recognition. Unpublished PhD Thesis, Psychology Department, University of Nottingham, Nottingham UK.
6. SpamBayes Open Source Spam Filtering Package <http://spambayes.sourceforge.net/>
7. SpamBayes Background <http://spambayes.sourceforge.net/background.html>
8. Wikipedia: Bayesian spam filtering: <http://en.wikipedia.org/wiki/Bayesian_spam_filtering>
9. WikipediaL e-mail spam: <http://en.wikipedia.org/wiki/Email_spam>
10. Wikipedia: The Blind Watchmaker <http://en.wikipedia.org/wiki/The_Blind_Watchmaker>