Jason Whitfield

Complex Systems 530 Project: Adaptive Search Engines

Purpose

The purpose of this project is to model the interactions of users with an Internet search engine. The Internet is composed of an incredibly large number of web pages, but search engines are able to find relevant results based on page content and popularity. So, order appears out of a complex system based on the actions of agents. An agent-based modeling approach applies especially well, since each user of the search engine is an agent that interacts with the environment of the websites and engine results. With this project, I examined how different page ranking strategies affect the relevance of results to agent's queries. To do this, I assessed how the average number of pages observed and read by users trended over time. I expected both to decrease as the search engine collected more data and was able to provide better informed results.

Literature Review

One paper I found was "Modeling and Simulation of Search Engine," which focuses on the mathematics behind matching queries to documents. Essentially, each document is made up of words, which can be grouped into terms. A query is also a term, and so a search engine matches terms between the query and its database of documents. The paper details how search engines can use multiple terms to restrict a large information space down to a small one. While this approach is equation based, it can still be useful in my model. I model users as agents, but I still need to find relevant results to each user's queries. So, this paper can help me match an agent's query to web pages and to other agents' queries.

Another paper, "A User Browsing Model to Predict Search Engine Click Data from Past Observations," is also mathematical, but similarly provides useful insight into how to implement my model. The paper looks at the probability that a user clicks a given link in the search results. This probability is affected not only by the relevance of the link, but also its placement in the list of results. This idea applies to my model, where agents must choose a link to click from the search engine results. I apply a probabilistic approach to the agent's choice in link and use this paper as a reference for the probability calculations.

Model Description

Environment

The environment in my model consists of the web pages, the user data collected by the search engine, and the search engine itself. To represent information contained in web pages, I use one-dimensional set of random integers. Since the information in a web page is usually centered on a certain topic, each page has an integer topic. The page data consists of integers normally distributed around this topic. This topic is examined by the user when choosing which page to click.

For the search engine to learn from the users, it must collect data about their actions. So, I define three pieces of data collected by the search engine. The first is the number of users that have visited a given web page. The second is the amount of information on the page that was consumed by users before they left. The third is how similar the page's topic was to their query. This page data will be stored along with the user's query, since the data will be more useful for similar queries in the future.

The search engine holds a list of web pages and the data collected from user actions. When a user makes a query, it orders this list using the data and a ranking strategy. Then, it observes the user's actions as they traverse the list.

Environment-Owned Variables

Web Pages
- topic
- page_information (set of info integers)

Web Page Action Data
- visits
- information_gained
- topic_similarity

Search Engine
- web_pages
- action_data

Environment-Owned Procedures

Search Engine
- order_pages(query)
- record_actions(user)

Agents

The agents of the system are the users of the search engine. Each user has some topic that they want to research. Like the web pages, this are be represented by an integer topic and a one-dimensional information set of integers normally distributed around the topic. They query the search engine about the topic and examine the resulting list of web pages. Then, they choose from the web pages based on a given page's placement in the list and its title's relevance to the information they are seeking. At each page, the user reads the page information and searches for the info that they are seeking. Any info that is found is removed from the search set, and the user updates their query to exclude that info. This continues until the user is satisfied with the amount of information that they have found.

Agent-Owned Variables

- topic
- sought_information (set of info integers)
- required_info_percent

Agent-Owned Procedures

- generate_query()
- choose_webpage(pages)
- read_webpage(page)

Topology

In the model, agents do not directly interact. Instead, they interact with the environment of the search engine. The data collected about agents affects subsequent search results, so the actions of each agent indirectly affect the actions of all future agents. The agents can interact with any web page, but as with real search engines, higher-ranked web pages are much more likely to be visited.

Initialization

There are a few global parameters I apply in my model. The first is the total number of web pages indexed by the search engine. The second is the range of possible information that can be found on web pages, i.e. the possible range of the integers in the info sets. The third is the length of the user and web page information sets. The last is the standard deviations of the normal distributions used by the pages and users.

The model is initialized by first generating the web pages. The web pages have totally random topics, independent of each other. I generate a large number of web pages so that users are most likely able to find the information they seek. Next, I initialize the user action data to be empty. Finally, I create the users iteratively. Each user is independent of all past and future users, so I generate them only when they are making their query.

Action Sequence

1. Set of web pages randomly generated, each with random topic and information vector.
2. New user generated with random topic and information vector.
3. User creates query based on sought information.
4. Search engine returns list of web pages based on query.
5. User chooses web page from list based on list placement and title relevance.
6. User reads web page until it gives up or reaches end of page.
7. User removes found information from their sought info vector.
8. Search engine records data about actions of user.
9. Repeat steps 3 through 7 until enough info has been found to satisfy user.
10. Repeat from step 2.

Query Generation

The query is representative of the information that the user still needs to find.

For a user u:

$$\text{query} = \text{mean}(u.\text{info})$$

Page Ranking

Each page must be scored based on user action data. The score is generated from that data and the search engine's ranking schemes. Then, the score is biased based on the similarity between the user's query and the data's query.

$$\text{scores} = \{0, \ldots, 0\}$$

For each data point d:

$$\begin{aligned} \text{score} &= \alpha W_0(d.\text{page\_click}) + \beta W_1(d.\text{topic\_similarity}) + \gamma W_2(d.\text{info\_read}) \\ \text{distance} &= |\text{query} - d.\text{query}| \\ \text{scores}_{d.\text{page}} &= \text{scores}_{d.\text{page}} + \frac{\text{score}}{\text{distance} + 1} \end{aligned}$$

Page Choosing

The user chooses a page based on its topics similarity to the query and its position in the list. Users generally prefer more similar topics and higher positions on the list.

$$\text{scores} = \{0, \ldots, 0\}$$

For each unvisited page p:

$$\begin{aligned} \text{distance} &= |\text{query} - p.\text{topic}| \\ \text{scores}_p &= \frac{1}{(\text{distance} + 1)(p.\text{rank})} \end{aligned}$$

Page Reading

The user then reads the page and removes found information from their search information. Also, the search engine must record their actions.

For a user u and page p:

$$\begin{aligned} d &= \text{new data} \\ d.\text{page\_click} &= 1 \\ d.\text{info\_read} &= |u.\text{info} \cap p.\text{info}| \\ d.\text{topic\_similarity} &= \frac{1}{|\text{query} - p.\text{topic}| + 1} \end{aligned}$$

$$u.\text{info} = u.\text{info} - p.\text{info}$$

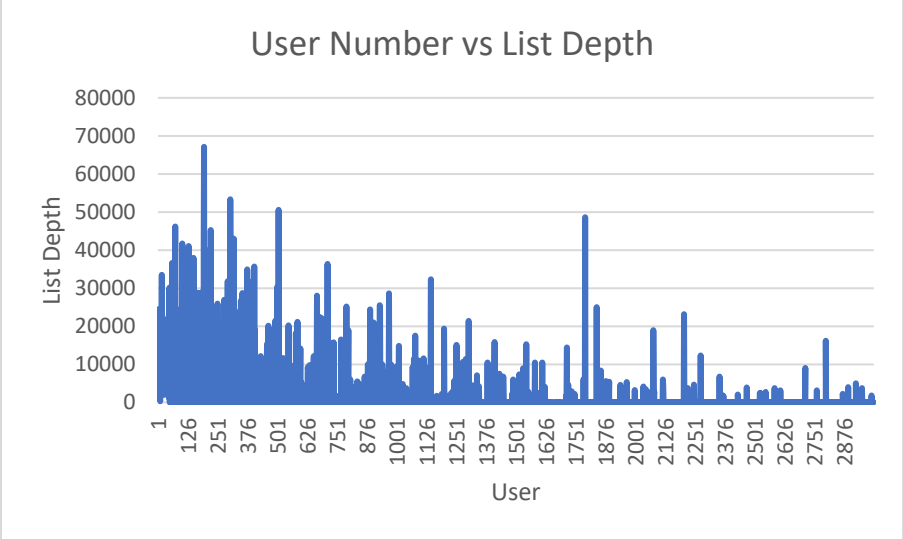
Assessment

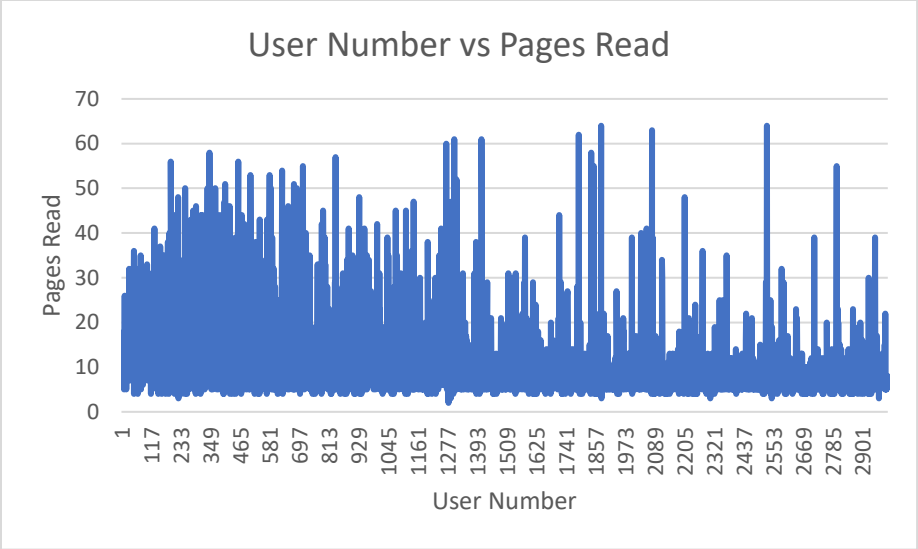
From this model, I hope to find out how different types of data and page ranking strategies affects the relevance of results to user queries. To measure this, I iterate over a very large number of users and measure the depth in the ranked web page list and the total number of web pages visited before each user is satisfied with the information they have found. I look at these as running averages and hopefully see them trend downwards over time.

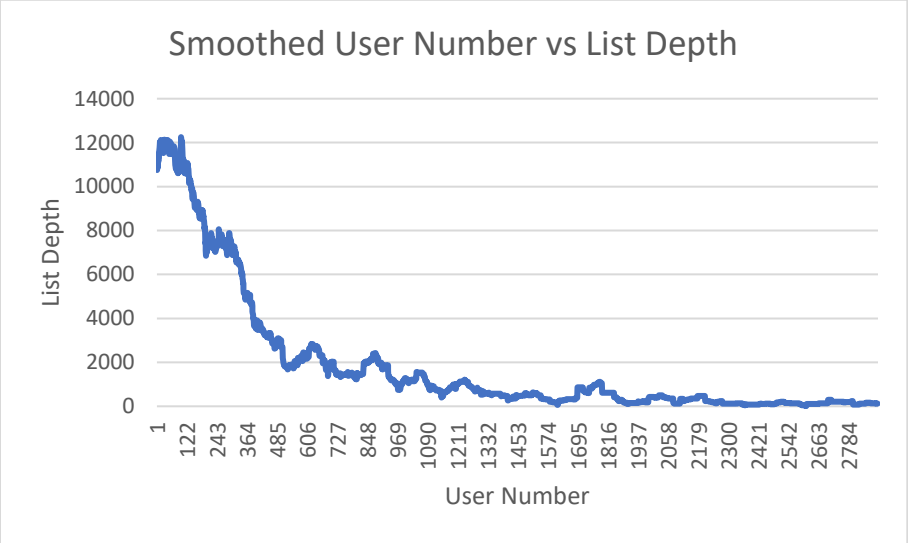
Results and Analysis

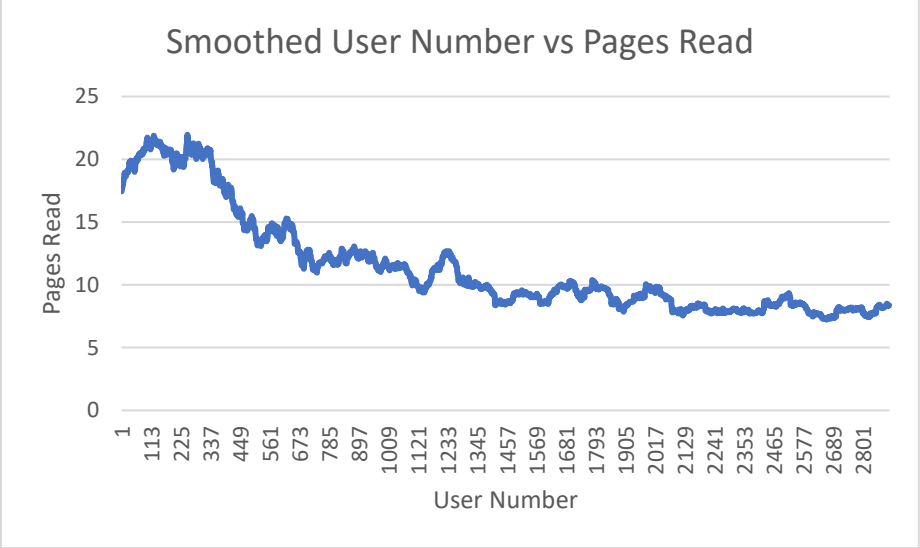
Model Run

The model runs generally as predicted. For each individual user, there is a lot of randomness to their page list depth and number of pages read, which makes sense given the randomness inherent in each iteration. However, over time, the average list depth and average number of pages read trend downwards as expected. The model is consistent in the trend of these averages, performing similarly over multiple runs at the same parameters. For 5 runs, I calculated the mean number of pages read for the last 500 users. The average of these means was 8.0456 with a standard deviation of 0.1767, showing good consistency between runs.

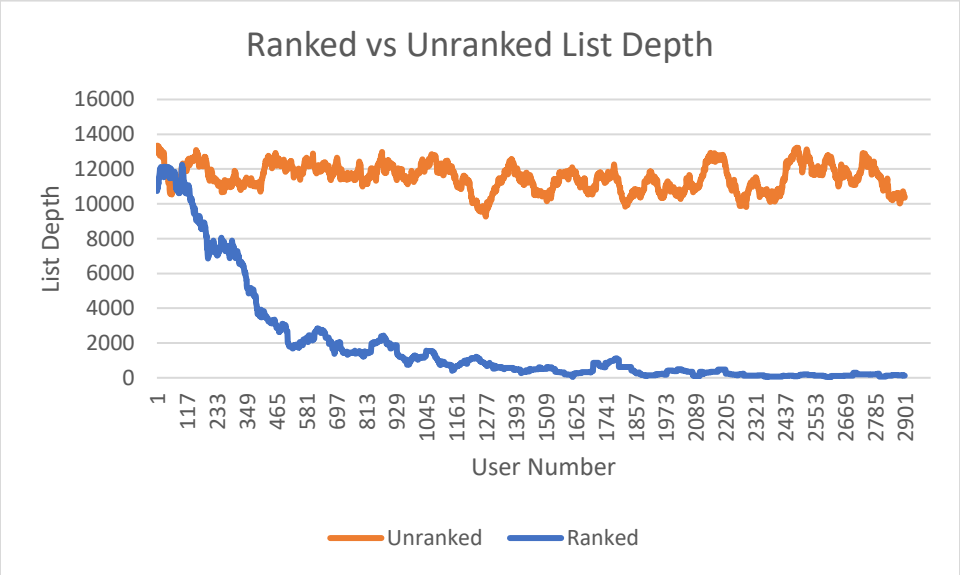


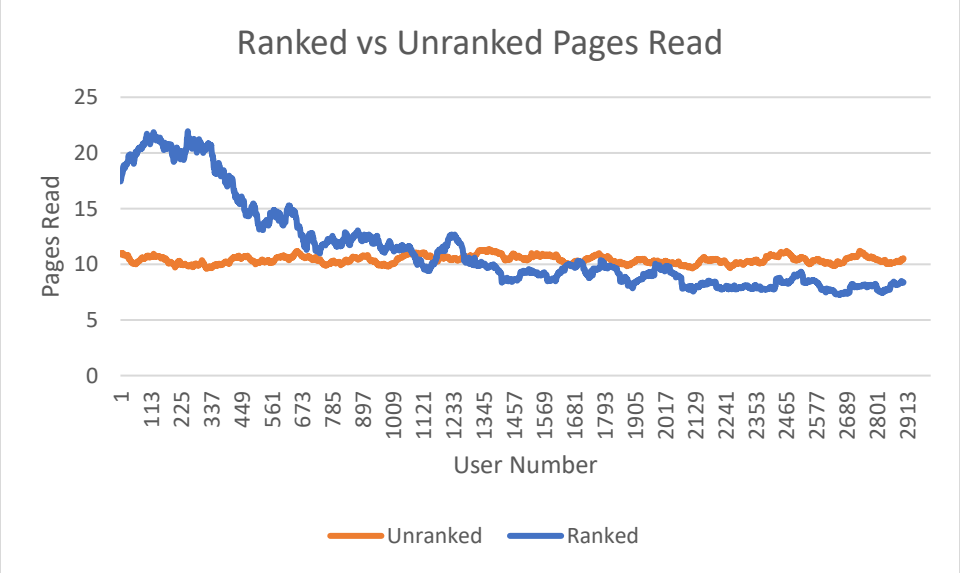

User Number vs Pages Read
Pages Read
70
60
50
40
30
20
10
0
1
117
233
349
465
581
697
813
929
1045
1161
1277
1393
1509
1625
1741
1857
1973
2089
2205
2321
2437
2553
2669
2785
2901
User Number


Smoothed User Number vs List Depth
List Depth
14000
12000
10000
8000
6000
4000
2000
0
1
122
243
364
485
606
727
848
969
1090
1211
1332
1453
1574
1695
1816
1937
2058
2179
2300
2421
2542
2663
2784
User Number


Smoothed User Number vs Pages Read
Pages Read
25
20
15
10
5
0
1
113
225
337
449
561
673
785
897
1009
1121
1233
1345
1457
1569
1681
1793
1905
2017
2129
2241
2353
2465
2577
2689
2801
User Number

The adaptive search engine is significantly better than an unadaptive, randomly ranked search engine. In one model run, over the last 500 users, the ranked search engine had a mean list depth of 168.1 and a mean number of pages read of 7.822. The unranked search engine had a mean list depth of 11771.9 and a mean number of pages read of 10.614. However, interestingly, the average number of pages read for the ranked search engine is initially higher than that of the unranked search engine. For the first 500 users, the ranked search engine had an average number of pages read of 18.228 versus the unranked search engine at 10.710. I suspect that this is because the ranked search engine will initially bring only a small subset of the pages to the front, and they are chosen by many users even if they are not the best pages for the users' queries. Over time, though, a more representative sample of pages are visited, and the search engine can bring relevant pages to the top of the list.





Heterogenous Users and Pages

I also explored the effect of having non-uniform users and web pages. To do so, for each user or web page, I uniformly distributed their parameters around their base value. Then, I ran simulations with no heterogeneity, only heterogenous users, only heterogenous pages, and both heterogenous users and pages. I found that heterogenous pages improved ranking efficiency while heterogenous users impaired it. This makes sense, as heterogenous users can lead to extremely picky users that are difficult to satisfy, but heterogenous pages can lead to very useful pages that the search engine can place near the top of search results.

	Pages Read Mean	Pages Read St. Dev.
Homogenous	8.138	5.481
Heterogenous Users	9.960	10.537
Heterogenous Pages	6.790	4.504
Heterogenous Both	8.084	8.596

Parameter Sweep

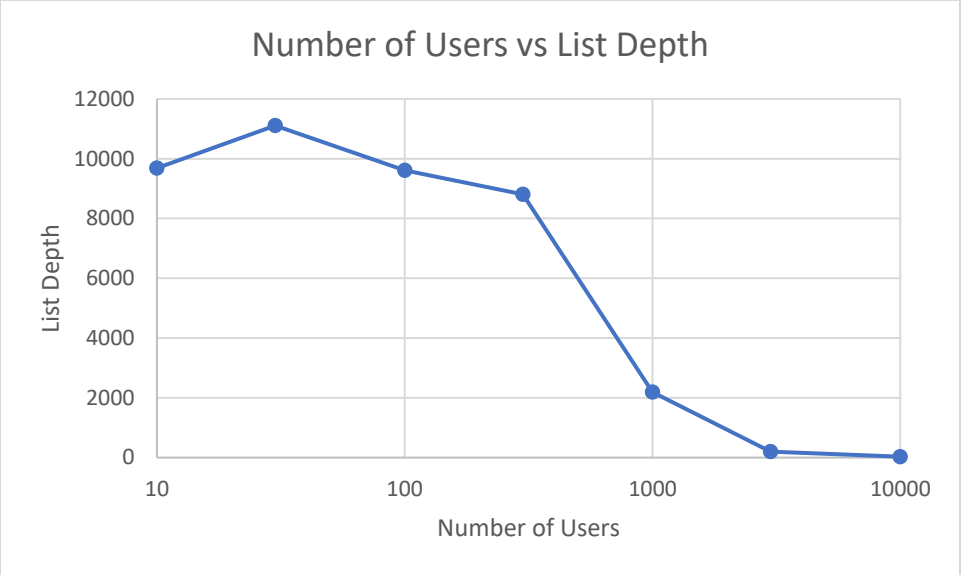
The model has the following nine parameters:

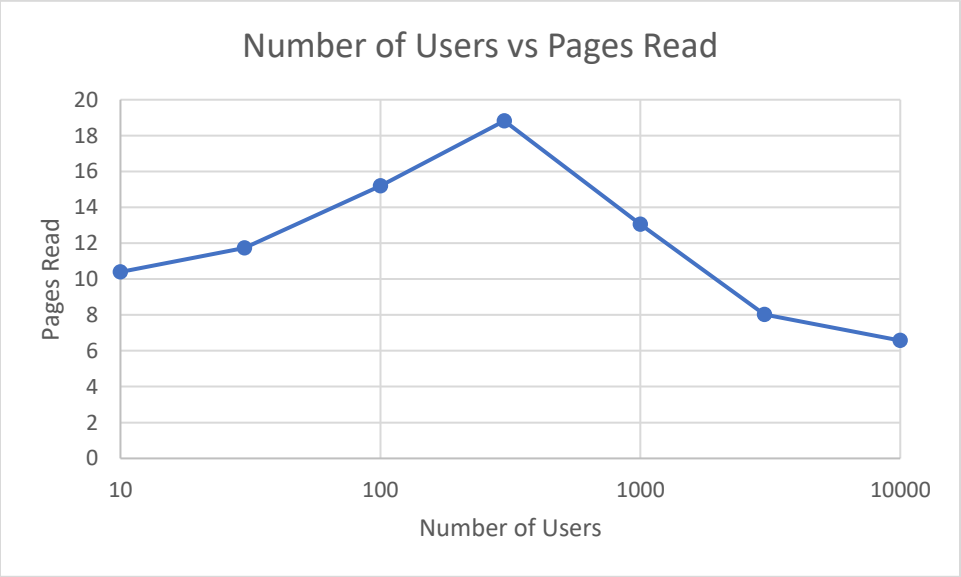
1. Number of users.
2. Number of web pages.
3. Maximum information value.
4. Page information length.
5. Page distribution standard deviation.
6. Search engine weights.
7. User information length.
8. User distribution standard deviation.
9. User satisfaction percentage.

Given the model's computation time, a continuous sweep through all parameters was not feasible. So, I explored the results of a few assignments to each parameter while fixing every other parameter. The results I examined were the average page list depth and average number of pages read over the last 500 users.

Number of Users

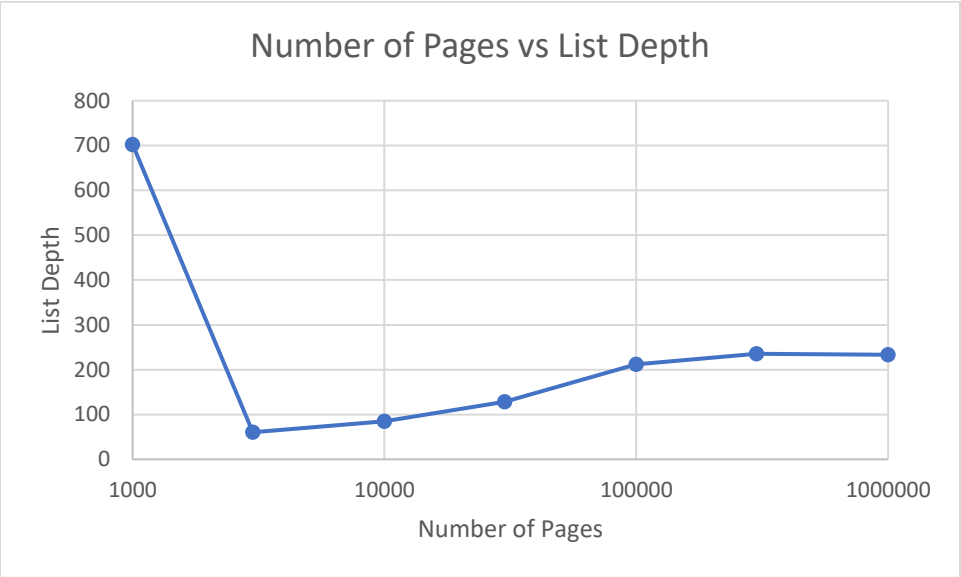
The average list depth trends downwards with the number of users, and then levels off. The average number of pages read trends upwards, then downwards, and then levels off. This is consistent with the trends observed for a single run of the model.

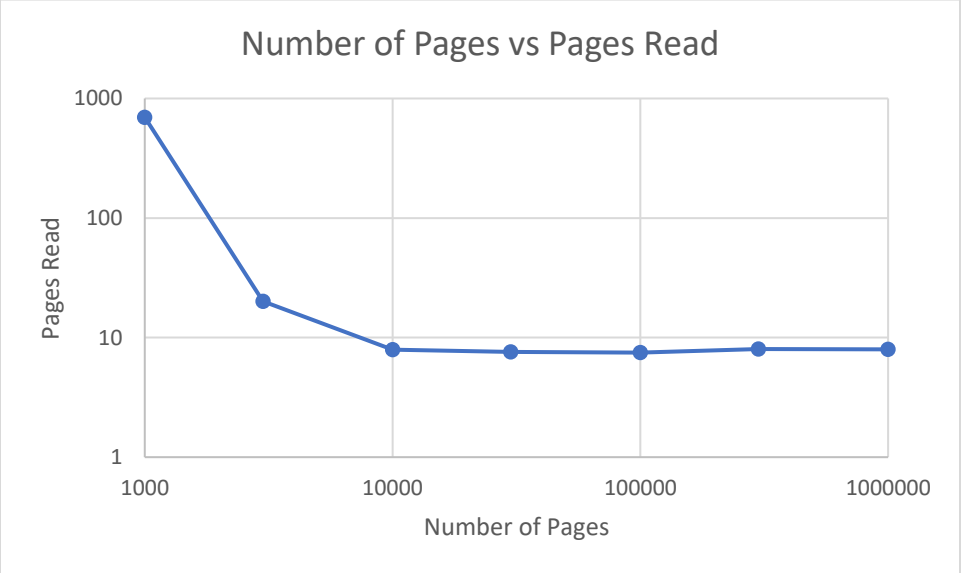




Number of Pages

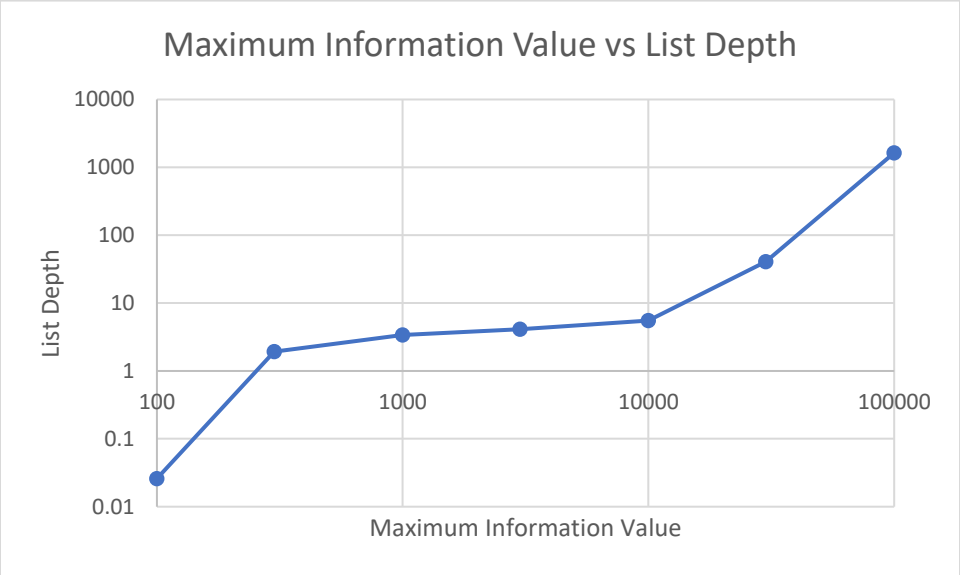
The list depth and pages read initially decrease when adding more pages. This makes sense, as new pages are adding more information to the system. Then, as the new pages add no new information, the values level off.

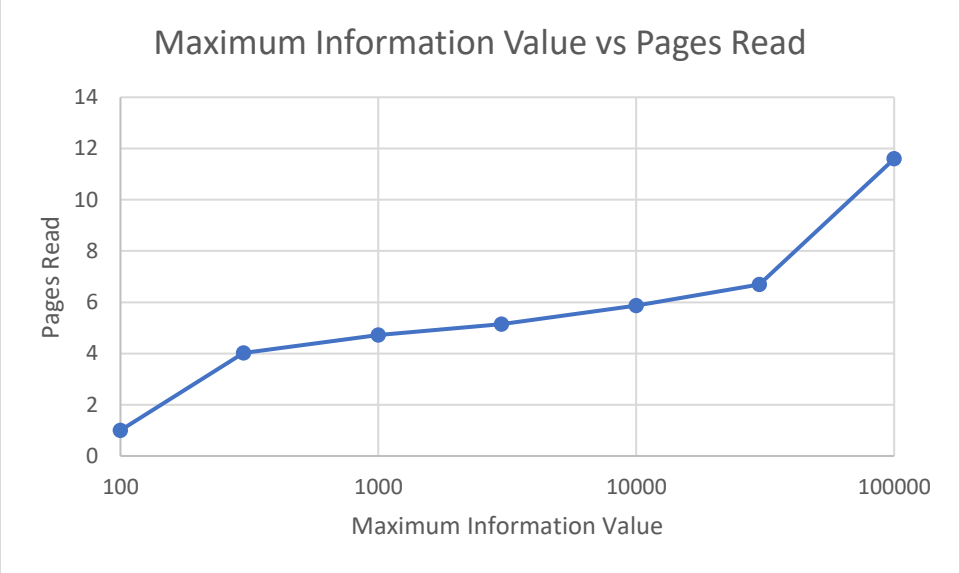




Maximum Information Value

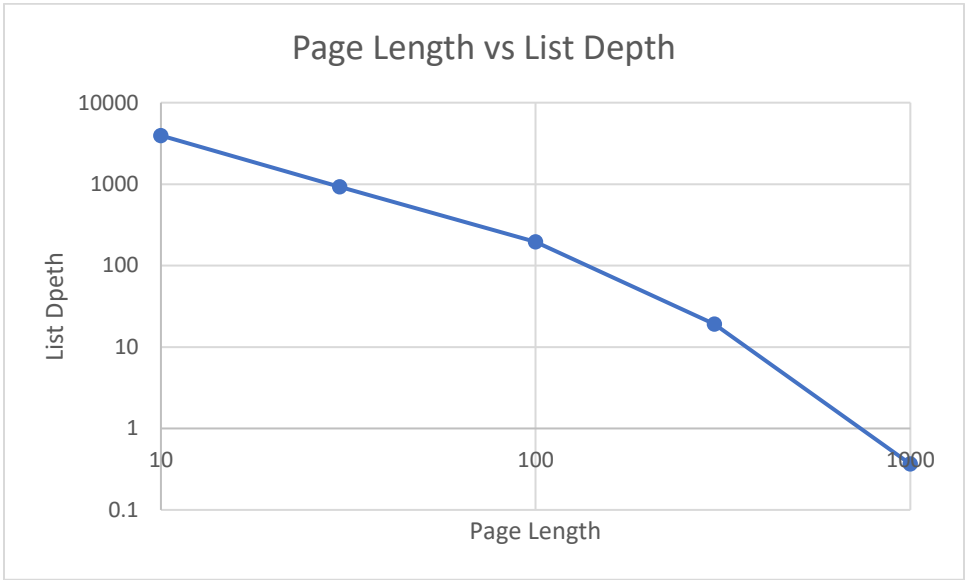
As expected, the list depth and pages read increase with the amount of information in the system. More global information means that a given random page is less likely to have useful information. I am unsure exactly why the values seem to level off between 300 and 30000 but change drastically before and afterwards.

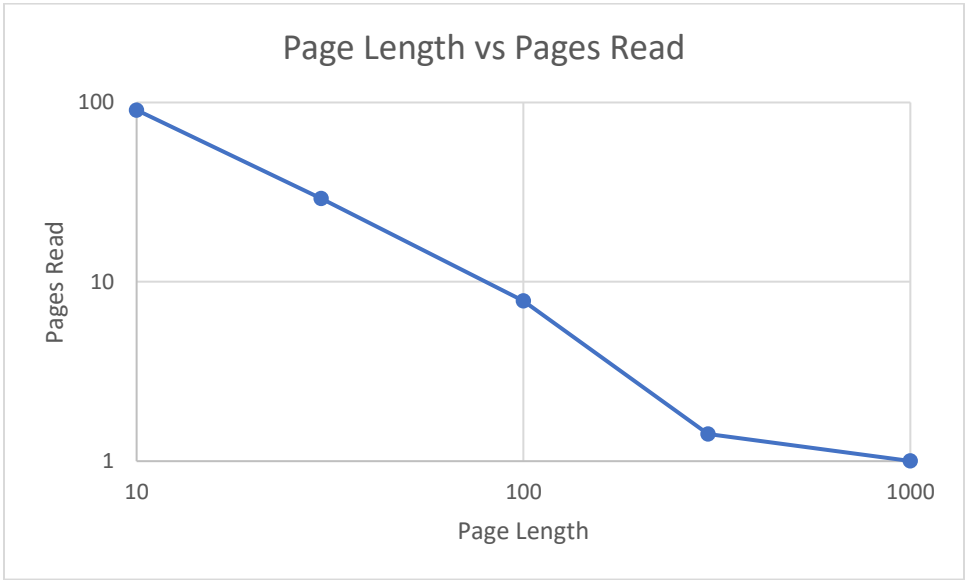




Page Length

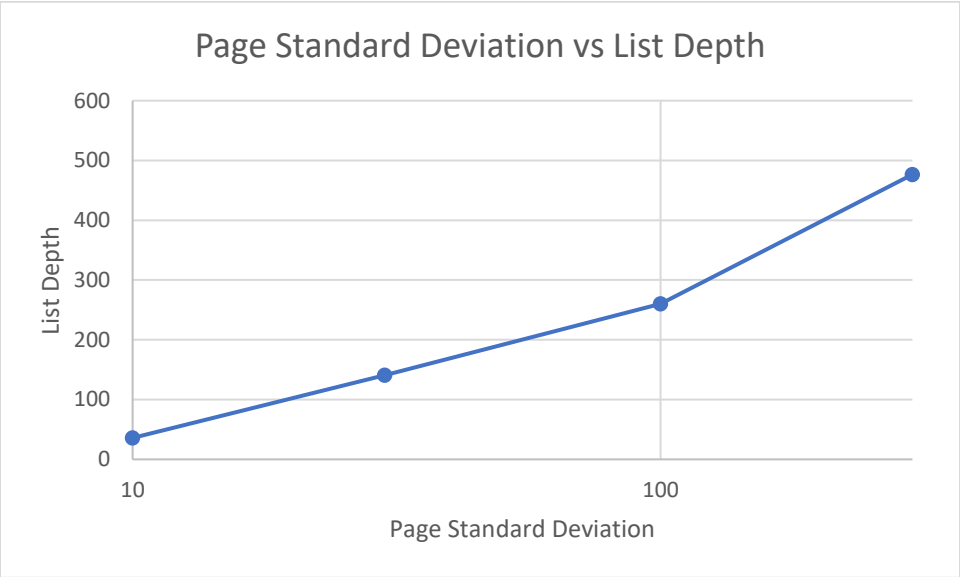
Both the list depth and pages read decrease when the page length is increased. Longer pages contain more information, so if the pages are longer, a user will not need to read or search as many pages.

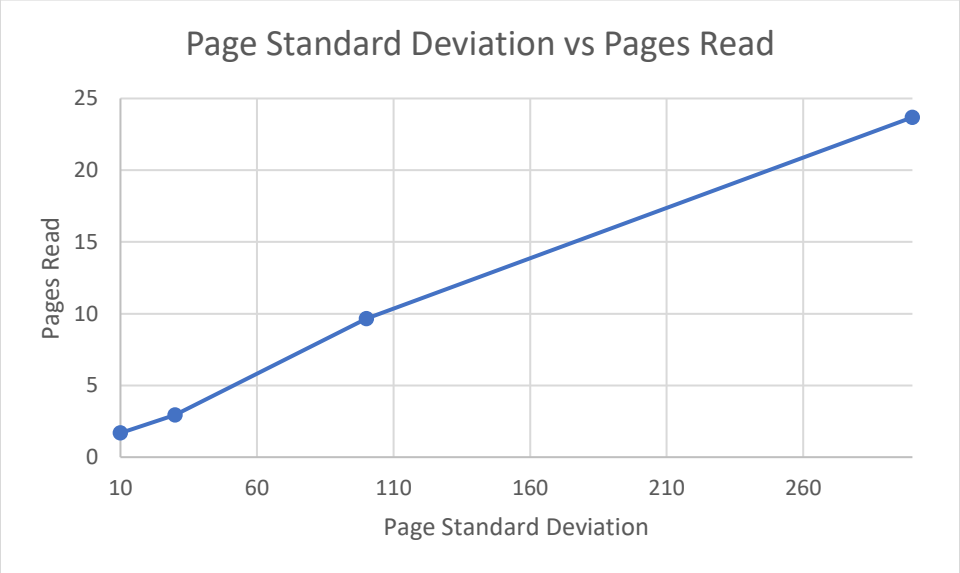




Page Standard Deviation

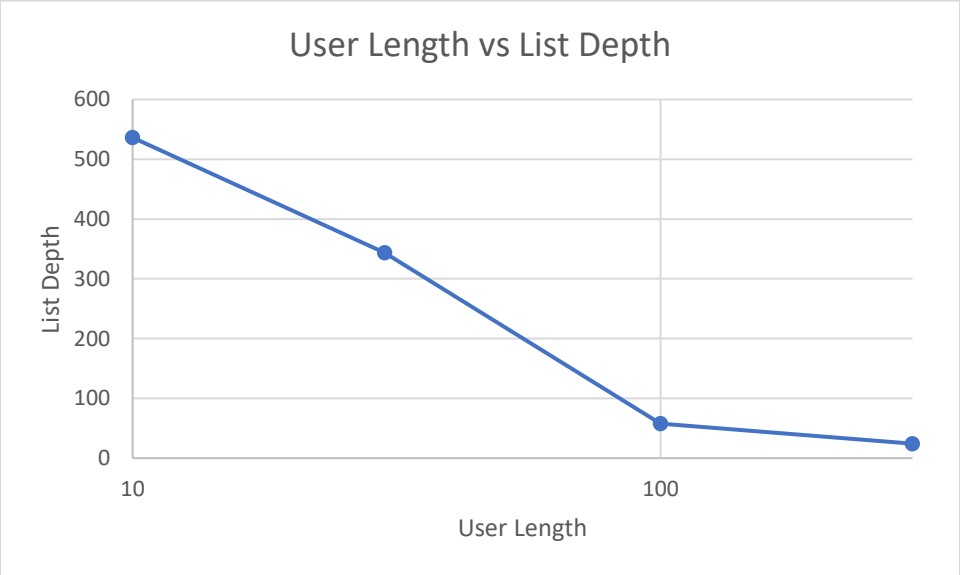
The list depth and pages read increase as the page information distribution's standard deviation increases. This is because a higher variance will cause the page's topic to be less descriptive of the information that the page actually contains.

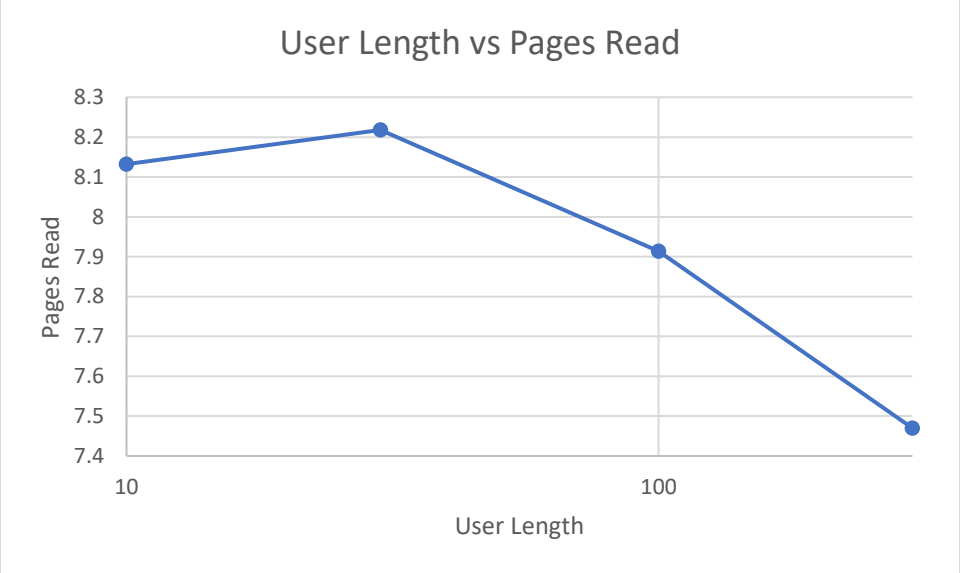




User Length

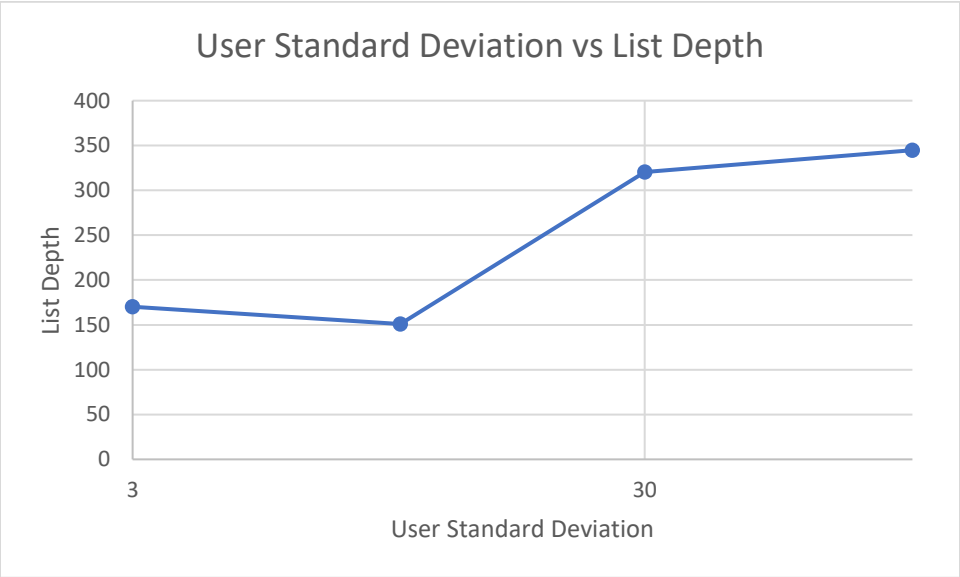
List depth and pages read trend downwards as user information length increases. This seems counterintuitive, as the user is seeking more information. However, since the user is satisfied after a proportion of their information is found, an increased sought information length gives them more flexibility in the information that they can find.

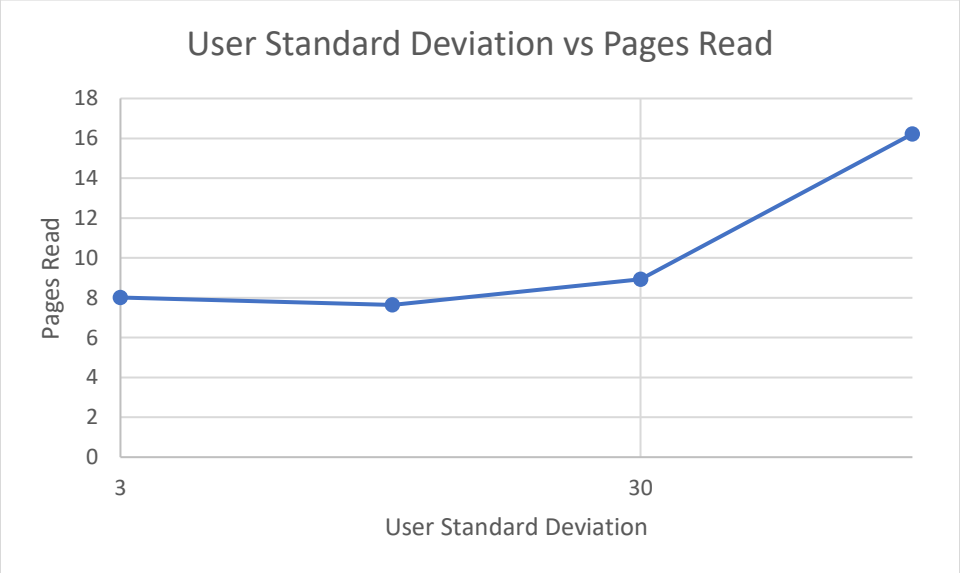




User Standard Deviation

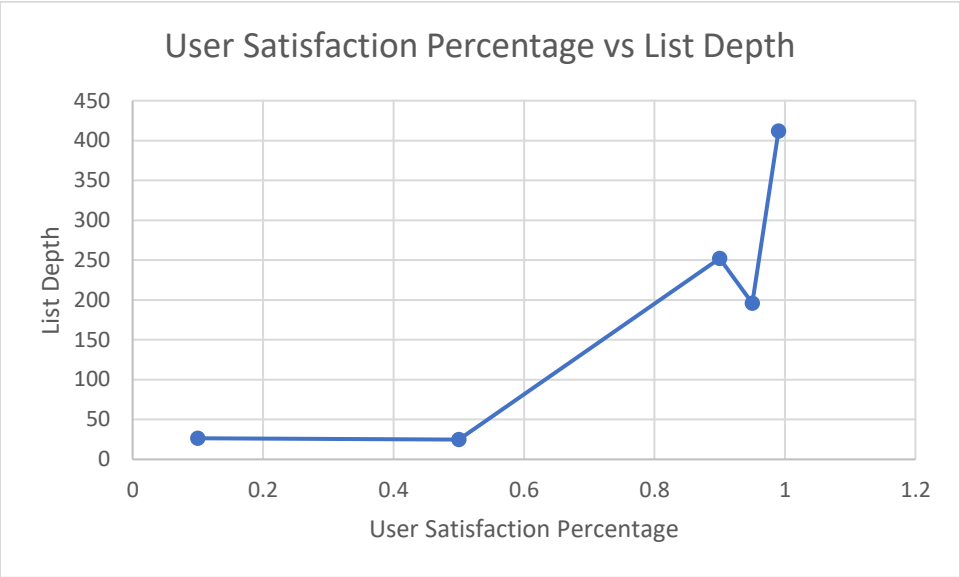
List depth and pages read increase as the user's information distribution standard deviation increases. This is because an increased standard deviation makes the user's queries less representative of the information that they are actually seeking.

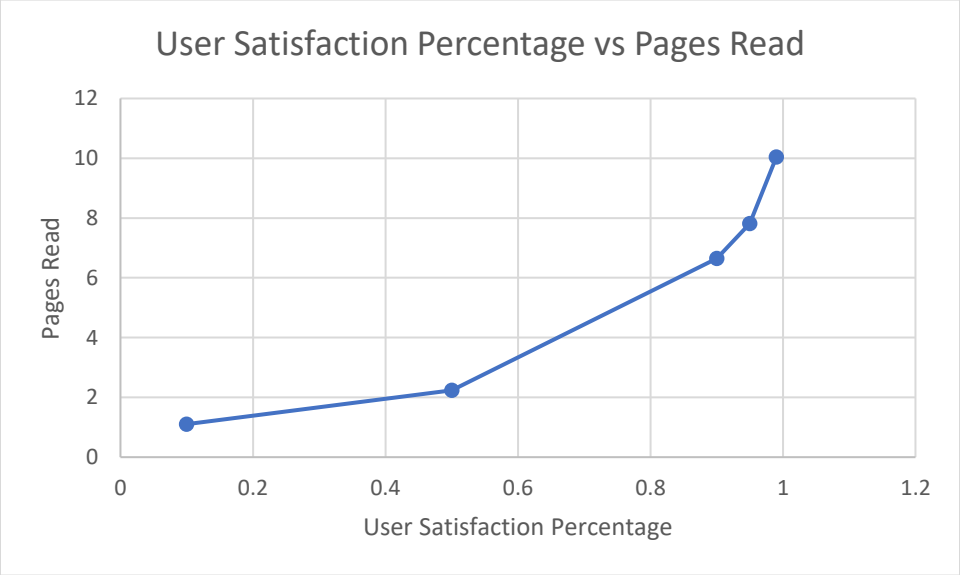




User Satisfaction Percentage

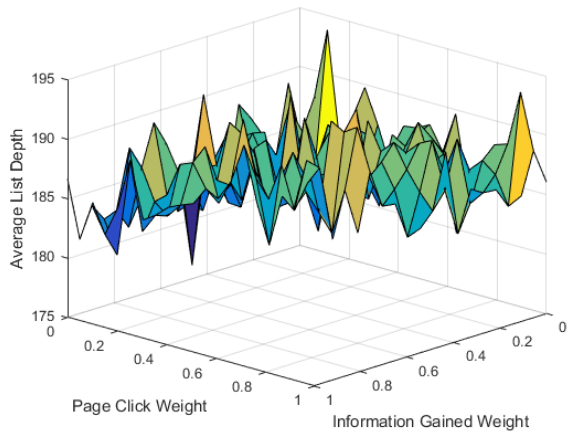
The list depth and pages read increase as the user's satisfaction percentage increases. This is obvious, as the user must find more information and has less flexibility about which information it can find.

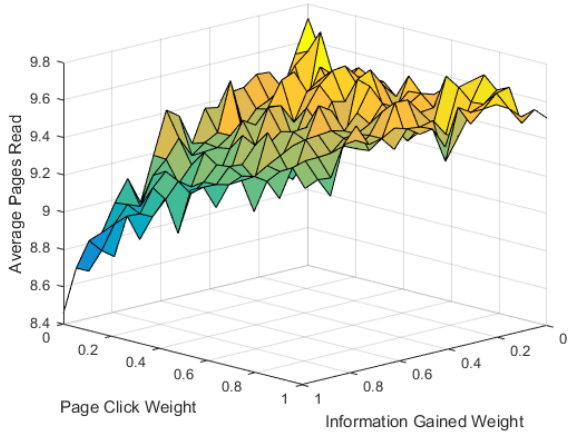




Search Engine Weights

There are three weights in the search engine's ranking scheme, but since the weights are normalized to sum to 1, the third weight is a function of the first two weights. So, I looked at the simulation results when modifying the first two weights. I found little correlation between the weights and the average list depth. All weighting schemes seemed to perform similarly. I suspect that this is because the types of data collected by the search engine are not independent. However, there was a strong correlation between the weights and the average number of pages read. It appears that the "information gained" data is more valuable than the other kinds of data, as the average number of pages read decreases as its weight tends to 1. I was interested in finding the true global optimum weighting, so I employed the Active Nonlinear Tests (ANTs) framework.

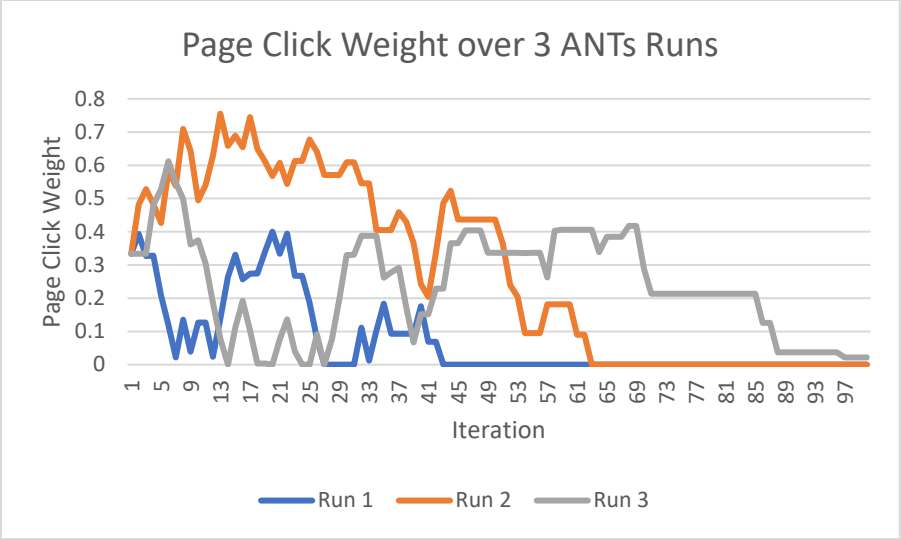


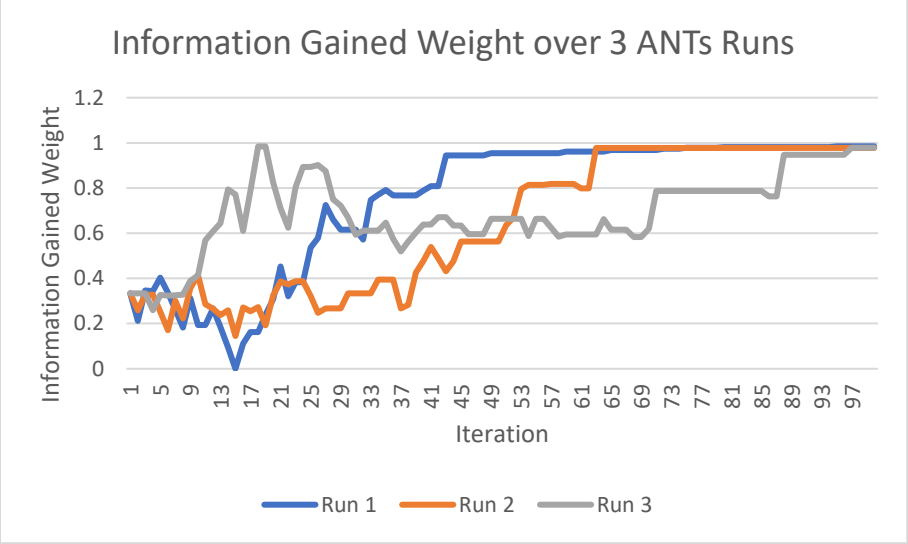


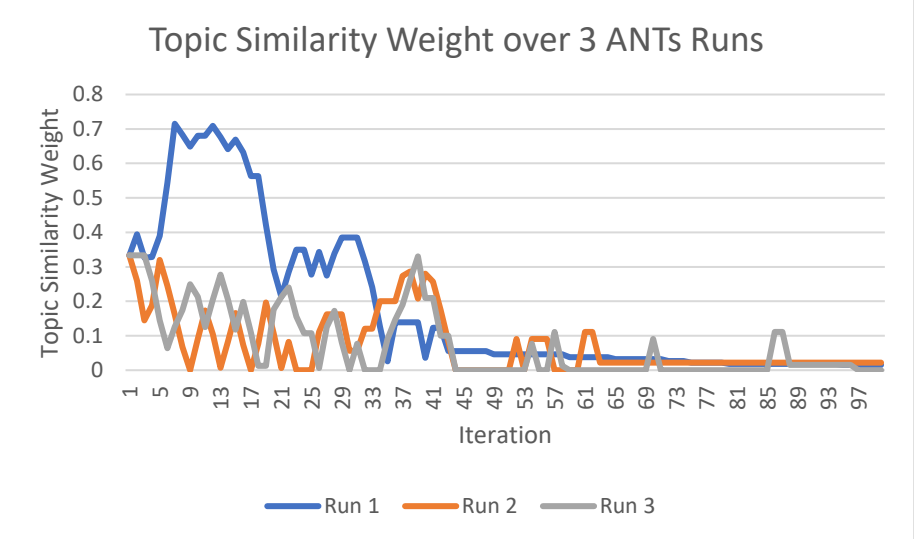
ANTs

I used ANTs to thoroughly search for the optimal weights. For the objective function, I used the average number of pages read by the user. If the number of pages read converges to a lower value, or converges faster, then the overall average will be lower. For the initial state and perturbations, I started with equal weights and randomly changed each by 0.1 in each timestep. I chose large perturbations because the randomness of the simulation makes it hard to tell if changes in the objective function are due to noise or real differences in the values of the weights. For the optimization method, I chose to use simulated annealing. Simulated annealing is similar to hill climbing but uses decreasing randomness to avoid local optima.

Using ANTs, I found that the true optimal weights are indeed those observed in the parameter sweep. I suspect that this weighting is optimal because "information read" is the only data that gives insight into the actual contents of the pages, so it is by far the most useful in ordering the relevance of pages. This result would be more interesting if there was other data that provided independent, useful observations, as there would most likely be a nontrivial optimal weight in that case.


Page Click Weight over 3 ANTs Runs
Page Click Weight
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Iteration
Run 1
Run 2
Run 3


Information Gained Weight over 3 ANTs Runs
Information Gained Weight
1.2
1
0.8
0.6
0.4
0.2
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Iteration
Run 1
Run 2
Run 3


Topic Similarity Weight over 3 ANTs Runs
Topic Similarity Weight
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Iteration
Run 1
Run 2
Run 3

Discussion and Future Work

Overall, I am satisfied with the performance of my model. I was able to observe the trends that I predicted, and I found some interesting interactions by playing with the parameter settings. However, there are places where the model could be improved. For one, I still don't fully understand the source of the extreme outliers in the data. I tried artificially reducing the information space that users could seek in order to prevent edge cases on the information distribution, but still observed outliers. I would like to reduce these outliers or at least understand their source. Also, the abstraction of information used in the model is very simplistic. It would be interesting to use a more sophisticated abstraction like bit strings, as they allow for more complicated behavior like partial matching. This would be a large undertaking, as all of the ranking math would need to be redone. Lastly, I was somewhat disappointed that the optimal weighting scheme was to fully weight one data type. This indicates that the data types that I chose are redundant. I would like to explore different kinds of data and hopefully find some that add novel information to the ranking and would lead to a more interesting weighting scheme.

References

Dupret, Georges E., and Benjamin Piwowarski. “A User Browsing Model to Predict Search Engine Click Data from Past Observations.” *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '08*, 2008, doi:10.1145/1390334.1390392.

Nasution, Mahyuddin K M. “Modelling and Simulation of Search Engine.” *Journal of Physics: Conference Series*, vol. 801, 2017, p. 012078., doi:10.1088/1742-6596/801/1/012078.