SEARCH ENGINE TUNING WITH GENETIC ALGORITHMS

by

Jeffrey Kyle Elser

A project submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2012

APPROVAL

of a project submitted by

Jeffrey Kyle Elser

This project has been read by each member of the project committee and has been found to be satisfactory regarding content, English usage, format, citation, bibliographic style, and consistency and is ready for submission to The Graduate School.

John Paxton

Approved for the Department of Computer Science

John Paxton

Approved for The Graduate School

Dr. Carl A. Fox

STATEMENT OF PERMISSION TO USE

In presenting this project in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this project by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this project in whole or in parts may be granted only by the copyright holder.

Jeffrey Kyle Elser

April 2012

ACKNOWLEDGEMENTS

I would like to thank Oracle RightNow and the Oracle RightNow applied research team for funding the research grant that made this work possible. I also appreciate being provided with access to relevant software.

All information contained within this work represents the views and opinions of the author and do not necessarily represent the views of Oracle RightNow.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF EQUATIONS

Equation Page

ABSTRACT

Tuning a local website to generate better local search results is a time consuming and tedious process. In this paper, a technique that can help to automate this process is described. Specifically, when a genetic algorithm is applied to a local search engine's weight parameters, the performance of the local search engine can be improved. Once good values for the search engine have been learned, it is easy to identify local Web pages that are candidates for further improvement.

Additionally, the same automated tuning method is applied to tuning a search engine that indexes structured knowledge base content. It is shown that knowledge base search weights can be tuned, resulting in improved search accuracy with regard to both result relevancy and ranking. The normalized discounted cumulative gain (nDCG) of tuned search results represents a nearly 25% improvement compared to the nDCG resulting from default search weights.

Finally a method of leveraging tuned search weights to further tune content is described. This results in the search engine overfitting to the content and providing very relevant search results for that point in time. The search tuning is repeated periodically when the weights no longer generalize and provide relevant results.

KEY WORDS

Genetic Algorithms, Search Engine Enhancement

INTRODUCTION

Search Tuning Problem

The Customer Relationship Management (CRM) software developed by Oracle RightNow includes many features designed to streamline customer interaction. One function of RightNow's CRM suite is a knowledge base where customers may ask questions that are answered by the software product rather than by company employees. The knowledge base has several main ways to accumulate knowledge: web indexing; structured, support agent-generated knowledge; customer community forums; etc. The RightNow product is able to index that content and allow customers to self-serve by searching for answers to their questions. [23]

The search mechanism includes some weighting parameters that allow a web master or knowledge manager to tune the search engine. Search engines must be tuned periodically in order to maximize the relevancy of the returned results. This work will explore methods of automatically optimizing a search engine's parameters for web indexed content and structured, support agent-generated knowledge.

Web Indexing

Web indexing involves spidering and indexing a company's website, allowing pre-existing information to be utilized. [8], [9], [23] A fairly standard approach for spidering and indexing the website is used. When a company installs RightNow's software, a service tech performs the configuration and spiders the company's website. Initially, a list

containing the location of words found on the page is built. These words can be found in the normal text body, HTML tags (e.g. H1), meta-tags and even URLs. Finally, an index is created from this list using the software's system of weighting.

Unfortunately, a company's Web pages are often not optimized for local searching. For example, a company might place its name in the description meta-tag of every page on its website in order to rank its pages higher than its competitors' pages on Internet-wide search engine rankings. However, this homogeneous local structure might reduce a local search engine's ability to differentiate between the content of local pages.

Because a company is in control of both its website and its local search engine, it should be possible to have its local search engine perform more accurately. As it may be very labor intensive for the company to manually update and optimize each Web page, a more reasonable request is to ask the local webmaster to alter a small subset of the website based on explicit instructions.

Structured Knowledge

Customer Care support agents have the ability to capture their troubleshooting steps and/or results in structured documents called Answers, which are stored in a knowledge base. The structure of these documents presents some additional features that can be leveraged for search engine tuning. For example, it is possible to create a hierarchical taxonomy of products or services, and specify a product or service in the Answer. That information can then be matched to a search stem to boost or filter search results. Unfortunately, this structure also creates some additional challenges – standard

web indexing does not work well on this type of content and additional search parameters are required.

Hypothesis

We hypothesize that search engine parameter tuning can be approached in two ways. First, the local search engine can be tuned using an automated process. Second, changes to the website that cannot be made through automation can then be suggested. The focus of this paper is on the first of these two approaches.

We hypothesize that by applying a genetic algorithm to tune the search weights that are applied to both web indexed content and structured content, search results can be improved in terms of both relevance and ranking compared to results presented using default search parameters.

BACKGROUND

Knowledge Management Overview

Knowledge management is the process of identifying, capturing, and reusing the information that results from day-to-day business in most industries. Although the idea of knowledge management has been a standard part of medium and large businesses for more than two decades [21], its processes and best practices are still being explored and improved upon today. This evolution of process is largely due to changing technologies involved in both the production and consumption of knowledge.

Webpage Content

One important type of knowledge is found in most companies' web pages. These pages are often created by the marketing department, but can be leveraged throughout the organization. This content often includes formal product documentation, how-to articles, and best practice guides. Frequently questions submitted to the customer care department can be answered by the documents created by the marketing department, and it is costly to ignore or duplicate that content.

The challenge with web content involves finding the content, organizing it, and presenting it in a timely and meaningful way. Finding and organizing the content is easily achieved with a standard local search engine. The challenging task is presenting the right content at the right time. There are two main use cases to consider: customer self-service and support agent reference.

First, most organizations will want to allow their customer base to self-serve as frequently as possible. Allowing the customer to find their own answers creates a better experience for the customer by providing the solution quicker and easier than if they had to contact and work with a live support agent. Allowing self-service also saves the organization money because they will not have support agents devoted to answering those questions.

Second, customer care support agents will leverage existing web content to supplement their own knowledge and solve service requests faster. Again, this creates a better customer experience and saves the company money.

Structured Knowledge

The second type of knowledge common throughout most industries is generated by customer care agents and stored in structured documents called Answers. Please see Appendix A for an example Answer.

Like web content, the primary way Answers are utilized is through customer self-service. Customers can search or browse existing Answers before submitting a service request. Again, this practice leads to increased customer satisfaction and reduced support costs.

If a customer cannot find the answer to their question and submits a service request, the customer care agent that handles the service request will start by searching existing Answers to make sure one does not already exist. If they cannot find a preexisting Answer, they immediately create a new one and start populating it with information as they troubleshoot the service request. Eventually, when they resolve the

service request, they will publish their new Answer, allowing other customers and support agents to benefit from that knowledge.

As with web content, it should be obvious that the success of the Answer publishing and reuse process hinges on being able to find relevant content. In other words, the searching mechanism must work very well.

Current Solutions

At present, all local search engines rely on manual tuning, either by adjusting weighting factors in the search engine itself, or by altering the content that is being indexed and searched. In the next three sections we discuss the relevant weights that can be tuned for webpage and Answer searching, as well as a recommended strategy for tuning the content.

Because it amounts to a combinatorial optimization problem, tuning the search weights manually is not typically very successful. Additionally, because the indexed content changes over time, this type of tuning has to take place periodically in order to adapt to the new content.

Webpage Indexing

The search engine used in RightNow's CRM product spiders and indexes a website through common spidering methods. The important feature of this search engine is the system of weights used to create the index. These weights act as multiplying factors when the score of a search term's occurrence in a document is calculated. For example, if

the search term is found in the title section of a Web page, and the title weight is 100, then the score of that term is multiplied by 100 and added to the document's score. The score for each document is thus the sum of the search term occurrences multiplied by their weights. The result of a search lists the pages based on their final scores in decreasing order. The first column in Table 1 contains a list of some of the standard weights and the second column contains their default values. The third column of Table 1 will be discussed later.

A majority of the weight identifiers in Table 1 refer to simple HTML or Meta tags such as <title> or <h1>. However, the meaning of a few of the tags may not be obvious and their functions are described below.

The meta-description identifier refers to a search engine specific Meta tag. This tag is only understandable to RightNow Technologies' search engine. It is not considered by most Internet-wide search engines and does not affect the page's appearance in a browser.

The multi-match weight is applied when more than one keyword occurs in a search. For example, if the search terms are "rocking" and "chair" and both are found in a document, the multi-match weight is applied to the document. This weight has no effect when the Boolean operator joining the two words is AND instead of OR.

The backlink weight is applied as a multiplier to the ratio of the number of links coming into a page versus the number of links going out. A page that has many outgoing links (such as a table of contents) will have its score reduced. A page that has many incoming links (such as one that contains important information) will have its score

enhanced. When a website has a common tree-like structure, a high backlink weight causes leaf node pages to be boosted in the returned results.

Weight Identifier	Default Value	Tuned Weights
backlink	1000.0	510.0
description	150.0	980.0
keywords	100.0	66.0
title	100.0	180.0
meta-description	50.0	920.0
heading 1	5.0	130.0
heading 2	4.0	340.0
heading 3	3.0	640.0
heading 4	1.0	720.0
heading 5	1.0	430.0
author	1.0	440.0
multi-match	1.0	170.0
text	1.0	0.0
url text	1.0	540.0
date	0.35	140.0
heading 6	0.0	0.0

Table 1: Search engine weights that are applied to web content. The first column is the weight identifier in the search engine, the second column is the default, pre-optimized value, and the third column is an example of the tuned weights generated by the genetic algorithm.

Answer Indexing

Although there is no need to spider Answers (they are generated within the system and effectively indexed when they are created), the Oracle RightNow products still maintain an index of Answers that is searchable. [23] Like web content searching, Answer searching utilizes a series of weights that act as multipliers against the term score. Table 2 contains a list of the search weights (weights 1-7) used in structured knowledge searching. Weight 8, usage boost, is a multiplier that is applied to the entire document score based on the how frequently the Answer is used.

	Weight Identifier	Default Value	Range Allowed[1]	Best Test Case Results
1	attachment	4	0-100	0
2	body	4	0-100	5
3	category	50	0-100	59
4	question	30	0-100	34
5	keywords	50	0-100	68
6	product	50	0-100	52
7	subject	45	0-100	40
8	usage boost	0	0-99	0

Table 2: Search engine weights that are applied to structured knowledge base content. The first column is the weight identifier in the search engine, the second column is the default, pre-optimized value, the third column is the allowed range the values can take on, and the fourth column is an example of the optimized weights generated by the genetic algorithm.

Content Tuning

Because manually optimizing the search weights is a combinatorial optimization problem and usually not very successful, most knowledge managers tend to opt for optimizing content. Even after tuning the search weights automatically, some additional content optimization will be required. In this section, we describe Oracle RightNow's best practice strategy for tuning content. [23] The purpose of describing this process here is to illustrate how time consuming and tedious optimizing the content can be, and how important it is to minimize the necessity for this effort. Additionally, it is worth noting again that because the underlying content changes over time, content tuning has to be performed regularly, typically every three months.

Oracle RightNow's content tuning process is based on the premise that for a given search term, the average rank viewed will be as close to 1 as possible. [23] Average rank

[1] These are the minimum and maximum values allowed by the Oracle RightNow products. They were not empirically proven to represent the optimal range.

viewed is defined as the average rank of the search results that are clicked. This means that the most relevant answer must be the first search result. Using a measure like discounted cumulative gain [26] would be more robust than average rank, but average rank is more intuitive for the knowledge managers that are using the process.

Oracle RightNow's Content Tuning Process:

1. Generate search stem list and sort by the number of visits, in descending order (Table 3)
2. For the top N search stems
    a. If (Average Rank Viewed <= 2), do nothing.
    b. If (2 < Average Rank Viewed <= 5), gray area; come back to these after handling all cases of type c and d.
    c. If (Average Rank Viewed == 0), no content exists; create new Answers for that search stem.
    d. If (Average Rank Viewed > 5)
        i. Generate histogram showing which search results get clicked most often (Figure 1).
        ii. For each result that gets clicked frequently, but is ranked lower than less popular results, adjust content to include more instances of the search stem.
        iii. Repeat steps i and ii until the histogram is sorted (Figure 2).

	Search Stem	Visits	Average Rank Viewed
1	UNINSTAL	131	1.35
2	FACEBOOK	65	2.92
3	31	54	1.09
4	UPGRAD	54	5.75
5	GUID ASSIST	52	1.72
6	SURVEY	51	3.49
7	ANALYT INCID ID	46	0.00
8	SMART ASSIST	46	3.05
9	INSTAL	41	11.31
10	CHAT	37	16.36
11	CLOUD MONITOR	37	2.60
12	PTA	37	3.56
13	WEBDAV	36	1.53
14	SLA	30	4.03
15	CUSTOM OBJECT	29	2.93
16	SERVIC PACK	29	2.40
17	CUSTOM PORTAL	28	4.48
18	API	27	3.03
19	SSL	27	2.85
20	RNTINFO	25	1.00

Table 3: Sample list of most frequently used search stems. Column 2 is the search stem. Column 3 is the number of times the search stem has been used in the last month. Column 4 is the average rank of the result that was clicked.

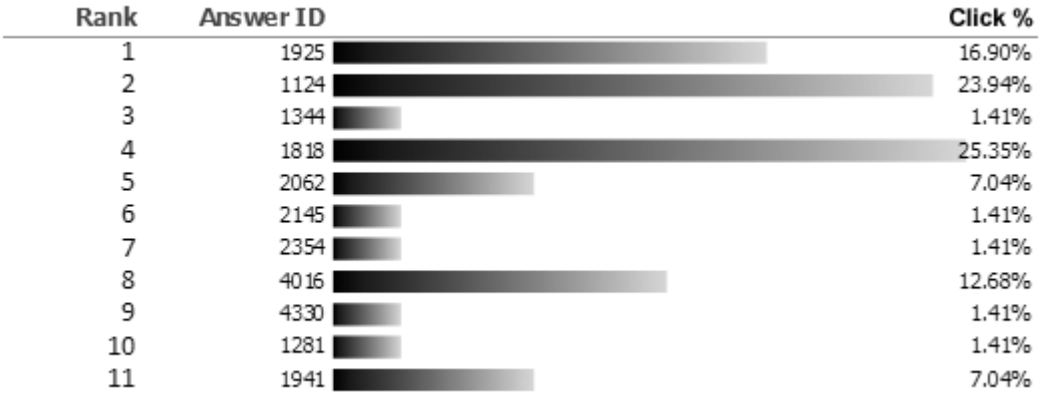


Figure 1: Sample histogram for a given search stem. Each row shows the percent of searches that click the result at that rank.

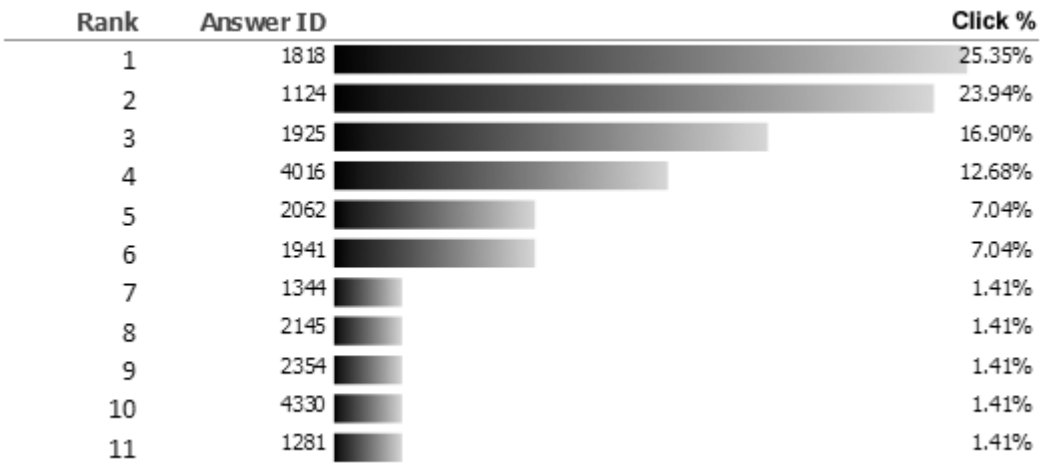


Figure 2: Sample histogram for a given search stem. Each row shows the percent of searches that click the result at that rank. This figure shows the optimal search results that are achieved after optimizing content.

Genetic Algorithm Background

A genetic algorithm (GA) is used to optimize the search engine parameters to achieve better rankings within the local website automatically. For the web content experiment, GAlib's prewritten functions were utilized [11]. GAlib is a collection of genetic algorithm functions written in C++.

For the structured content experiment, it was not possible to install a preexisting genetic algorithm package on the server running the Oracle RightNow software. It was necessary to construct a GA from scratch using the python programming language.

A variant of Goldberg's Simple GA [12] is used in the web content experiment. The GA used in the structured content experiment is based on the canonical GA

described by M. Mitchell in the first chapter of An Introduction to Genetic Algorithms [20]. The main elements of the simple GA are standard mutation, standard crossover, and non-overlapping populations. The fitness function will be explained in Section 3.

Below is an overview of the genetic algorithm used in all experiments:

1. Randomly generate a population of *n* individuals; each individual consists of a list of either 16 real numbers (web content experiment) or 8 integers (structured content experiment)
2. Loop *number_of_generations* times
    a. Calculate fitness for each individual
    b. Loop $n/2$ times
        i. Perform fitness proportional selection to select two individuals. We use the roulette wheel selection method in all experiments.
        ii. Perform single point crossover based on the crossover probability
        iii. Perform gene-wise mutation on each individual based on the mutation probability
        iv. Add the two individuals to the new population

APPROACH

Web Indexing Experiment

To produce better search engine rankings, the feature weights in the search engine's configuration file need to be improved. Figure 3 shows the proposed design for the program that will optimize web content search weights as well as suggest content changes. The Search Query, Set of Web Pages, and Desired Ranking in Search Results are all discussed in the Dataset section below. The Search Engine Update / Machine Learning component is then described in the Genetic Algorithm Configuration and Fitness Function sections.

Dataset

Initially the Webmaster must supply training data. He or she must identify the ranked pages that should result from a particular search query. This information is stored in a batch file. It is possible that user input can be used to create this batch file automatically [1], [22]. This is discussed further in the conclusions section.

The websites used for this experiment were created from 11 newsgroup articles. These articles were selected from a 20,000 article data set hosted on the UCI Knowledge Discovery in Databases Archive [13].

The articles were chosen for their structure and word count. Larger documents tend to work better in a search engine simply because they are more likely to contain

multiple instances of the search term. The 11 selected articles were formatted by hand to include HTML formatting tags and relevant Meta tags.

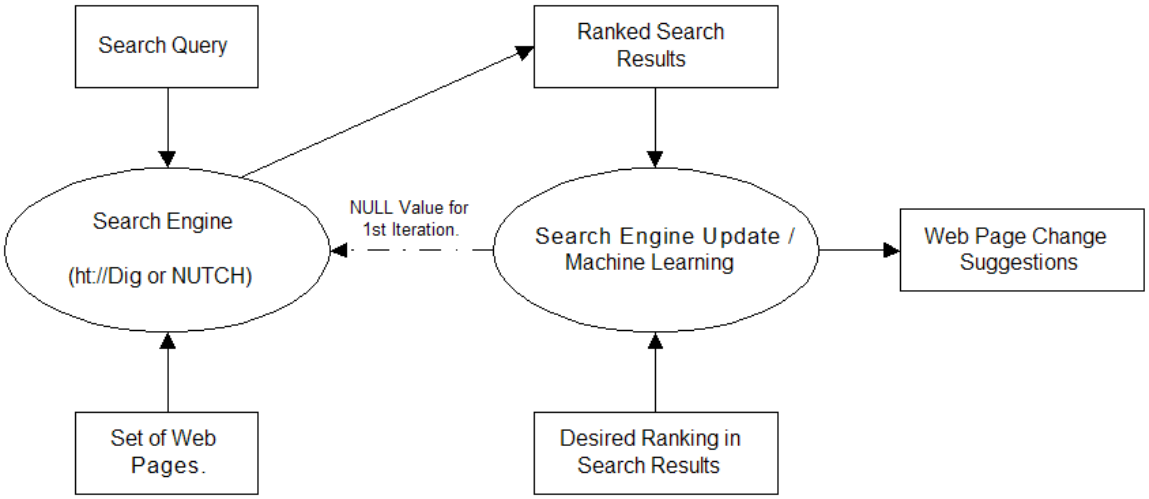


Figure 3: Diagram of the proposed web content tuning mechanism. A search query, set of web pages, and search engine parameters are provided to the search engine. The genetic algorithm is then applied to the search results and the desired rankings, resulting in new search engine parameters. When the GA finishes, new search engine parameters and content changes are suggested.

Genetic Algorithm Configuration

The GA begins by creating a random population of genomes. Two different population sizes, 1000 and 10000 were tested but the results were almost identical. All the data in this paper was collected from tests using populations of size 1000. For the problem at hand, the genome contains 16 real numbers ranging from 0.0 to 1000.0. Each number corresponds to one of the weights shown in Table 1. Using random initial values is a common technique for reducing the number of generations required for a genetic

algorithm to converge upon an acceptable result. There is evidence that the quality of the random number generator affects performance as well, but for these experiments, only GAlib's built in random number generator was used [6], [7], [19].

The GA executes for a predetermined number of generations. Elitism is turned on to ensure that the fittest individual is retained from one generation to the next. The probability of mutation is set to 0.01 and the probability of crossover is set to 0.6, commonly used values. [20]

GA Parameter	**Value**
Population Size	1000
Mutation Probability	1%
Crossover Probability	60%
Number of Generations	100

Table 4: GA Parameters used for the genetic algorithm in the web content experiment.

Fitness Function

A fitness function is required for all GAs. The fitness function for this problem is a distance measure between the top ten actual rankings of the Web pages and the top ten desired rankings from the batch file (if there are that many). Equation 1 defines the fitness function where D is the absolute value of the difference between a page's actual ranking and its desired ranking. For example if a page's actual ranking is 5 and its desired ranking is 2, then the distance is 3. In the batch file, if there are fewer than 10 desired rankings, then the unspecified positions are considered to match perfectly.

$$F = \frac{1}{\sum D + 101}$$

Equation 1: Fitness function used in the web content experiment. D denotes the absolute value of the difference between a given page's actual ranking and its desired ranking.

There are two special cases for the fitness function. First, if the page is not included in the top ten results in the actual rankings, but it should be according to the desired rankings, then D is set to 100 for that page. Second, if the actual ranking matches the desired ranking, then D is set to -10 for that page. This defines the largest possible distance to be 1000 and the smallest distance as -100. Substituting those values into the fitness function, the largest distance gives a fitness value of 1/1101 or 0.0009, and the smallest distance gives a perfect fitness value of 1. The addition of a penalty (+100) and reward (-10) based on placement within the ranted results accommodates a positional user model [3] where results on the first page are much more likely to be clicked.

Structured Knowledge Experiment

Dataset

The dataset for the structured knowledge experiment consisted of more than 1400 Answers from Oracle RightNow's product site. Instead of specifying the optimal search results by hand, we mine millions of rows of clickstream data to find search stem and clickthrough data. In order to reduce noise and computation time, we select the top 82

search stems. We then use the clickstream's clickthrough data to create a sorted histogram for each search stem, similar to the one shown in Figure 2.

In Figure 4 we see that the search volume for most search stems is relatively low. There is also a certain amount of noise in the clickstream. Users will occasionally perform a search and then get distracted or change interests. The next answer they view might then be completely unrelated to the search. To avoid this type of noise, we use only search stems that have had clickthrough to the same Answer several times. For this reason, search stems past 82 were discarded.

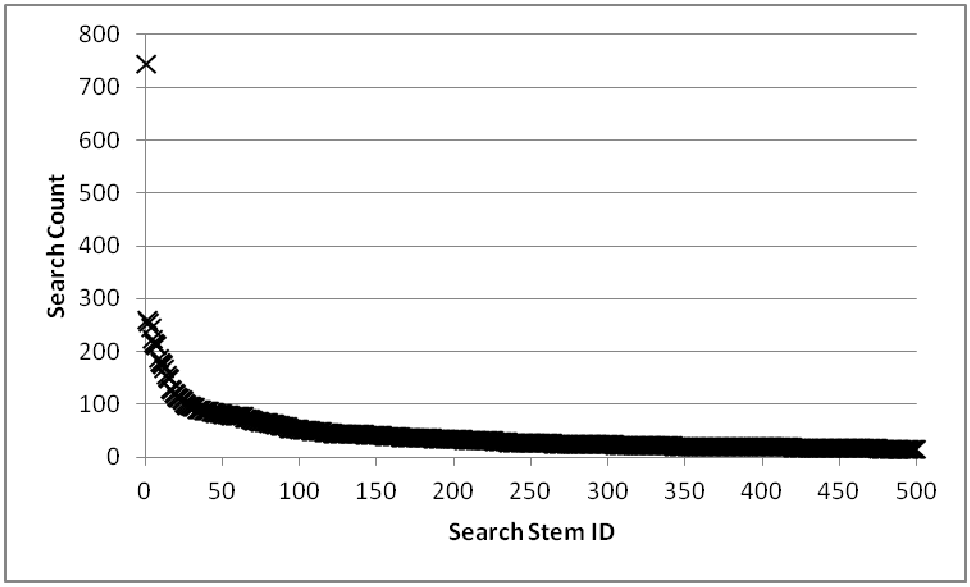


Figure 4: Search count for each search stem found in the clickstream.

Although it was much more difficult to gain access to this data, using real world data has a few important benefits. The most obvious is that the previously theoretical idea of using a GA to tune the search has been proven to work in a real world situation.

Additionally, by using a much larger set of optimal histograms in the fitness functions, a single change in rank in the results for any given search stem will have a lower impact on the fitness score. This smoothes the fitness function, making the search space easier to traverse. Finally, because there is no longer a need to pick the optimal search rankings by hand, this solution is much easier to implement.

Genetic Algorithm Configuration

As in the web content experiment, we used the simple GA described in the Genetic Algorithms Background section. The parameters used are listed in Table 5 below. The genome consists of 8 integers, representing the 8 weights listed in Table 2.

GA Parameter	**Value**
Population Size	100
Mutation Probability	1%
Crossover Probability	60%
Number of Generations	1000

Table 5: GA Parameters used for the genetic algorithm in the structured content experiment.

Fitness Function

The Levenshtein Edit Distance. The Levenshtein Distance compares two strings to determine their similarity [18]. It counts the number of insertions, deletions, and swaps that need to be made to make two strings identical. For example, the strings “FROM” and “FARM” have a Levenshtein Distance of two. They can be made identical by deleting the ‘O’ and inserting an ‘A’ in “FROM” or swapping the ‘R’ for an ‘A’ and the ‘O’ for an ‘R’.

We reduced our problem of finding the distance between two lists of URLs to finding the distance between two strings by treating each URL in the list as a character in a string. Then matching two URLs is synonymous with matching two characters in a string.

The problem with this new fitness function is that it only changes value when a URL moves into or out of the exactly correct position. For example, having 10 desired results all included in the top ten actual results but scrambled so that they are not in the correct position scores exactly the same as having the 10 desired results all fall to the very bottom of the list. Obviously having the 10 results in the top ten can be considered a very good solution because they will all be seen on the first page of the search results. Therefore it is important for the fitness function to distinguish between these two cases.

To augment the Levenshtein Distance, we use the un-weighted distance from the desired ranking to the actual ranking in the returned results list. We only call this part of the fitness function when a desired result does not match the actual. That avoids double scoring results that the Levenshtein Distance function already considered. Equation 2 shows this portion of the fitness function where Max refers to the largest possible distance and Δ refers to the distance between the actual result and the desired result. For lack of a better term, we will call this part of the function the rank distance.

$$\frac{\Delta}{Max}$$

Equation 2: The "rank distance" is the raw distance between a given set of results and their desired ranks. This is similar to D in Equation 1.

The Levenshtein Distance is added to the average of the rank distance for all the terms that did not match perfectly. Equation 3 shows the new fitness function in its entirety.

$$F = LD + Avg\left(\frac{\Delta}{Max}\right)$$

Equation 3: Fitness function used in the structured content experiment. LD denotes the Levenshtein Edit Distance for actual rankings and desired rankings. The second half of the equation is the "rank distance" that factors in the raw distance between a given set of results and their desired ranks.

Normalized Discounted Cumulative Gain. Normalized discounted cumulative gain (nDCG) is a common measure of result quality in many areas of information retrieval [26]. As seen in Equation 4, cumulative gain is the sum of the relevance measures for each result *r* through rank *n*. The problem with cumulative gain is that it ignores the rank of each result. So the cumulative gain for the optimal results would be the same as the cumulative gain for the optimal results in reverse order.

$$CG = \sum_{i=1}^{n} r_i$$

Equation 4: Cumulative Gain where n is the number of results to examine, i denotes the rank, and r_i is the relevance measure of the result at rank i.

Because the order of results is critical [3], especially when considering how results are paged (it is quite uncommon for a user to examine results past the first page), we used the discounted cumulative gain. In Equation 5, we see that DCG discounts the relevance of each rank based on the log of that rank [14].

$$DCG = r_1 \sum_{i=2}^{n} \frac{r_i}{\log(i)}$$

Equation 5: Discounted Cumulative Gain where n is the number of results to examine, i denotes the rank, and r_i is the relevance measure of the result at rank i.

It is important to account for the variance in the number of results and the amount of traffic they receive. For example from Table 3, the search stem "UNINSTAL" received 131 visits, so the cumulative gain for those results would be 131. However, the stem "API" had only 27 visits, so its cumulative gain would be 27. Although it would be interesting to see how biasing the search engine tuning towards more popular search stems would affect user satisfaction, that effort is beyond the scope of this work. For our purposes, it is necessary to normalize the DCG so that all search stems get the same priority in the fitness function.

In Equation 6, the normalized discounted cumulative gain is the actual discounted cumulative gain divided by the ideal discounted cumulative gain. The ideal discounted cumulative gain is the discounted cumulative gain of the optimal results.

$$nDCG = \frac{DCG}{IDCG}$$

Equation 6: Normalized Discounted Cumulative Gain where DCG is the discounted cumulative gain of the current results and IDCG is the discounted cumulative gain of the optimal results.

Please review the section "Normalized Discounted Cumulative Gain vs. Expected Reciprocal Rank" in the discussion chapter for a deeper discussion on the pros and cons of using nDCG over other methods.

Because we are using 82 search stems, and summing their nDCG, we know that the best possible fitness for this data set is 82.

RESULTS

Web Indexing Experiment

Twelve tests of 1000 trials were performed on the eleven Web pages. The search query “introduction” was used throughout the tests. Several of the test pages were large introductory texts and FAQs. Each of these articles contained several occurrences of the query term “introduction” in a variety of HTML and Meta tags. The other pages were argument style posts that contained few if any occurrences of the query term.

The twelve tests were designed so that each one had a different desired ranking. The first test’s desired ranking was realistic in the sense that a real Webmaster chose it. The desired results for the other eleven tests were picked at random.

The tests performed show that some improvement in ranking is possible, although not guaranteed. In four of the twelve tests, including the realistic one, perfect rankings were achieved. In four other tests, the rankings were improved as the search engine weights were modified by the genetic algorithm. In the remaining four tests no improvement was attained beyond the initial random weighting. The average distance between the actual and the desired ranking for those last four tests was 5, which as shown in Figure 5, is a common initial distance. This eliminates lucky initial ordering as a factor for lack of improvement.

While 100% accuracy was achieved in some test cases, this level of accuracy was not always possible. The main reason for the GA to fail to achieve perfect rankings lies in

not allowing negative weighting[2]. For example, maintaining other factors constant, if one page has three occurrences of a keyword in its body text, while another page has only one occurrence of the keyword in its body text, then no value for the body text weight will cause the second page to be ranked ahead of the first page. The more pages that appear in the training file, the higher the chances are that there will be no perfect set of search engine weights.

In Figure 6, the average fitness values from the 12 tests are plotted against the number of generations. It is helpful to review Equation 1 before proceeding. The possible range of values from the fitness function is from 1 to 1/1101. However, it is necessary to realize that after excluding the perfect value of 1, the next best fitness value possible is 1/23 or 0.0435. The score 1/23 comes from 8 perfect matches (d is -10 for each) and the two remaining pages being out of order by just one position (d is 1 for each). For example, if the desired ranking was {5, 2, 1, 3, 4, 6, 7, 8, 9, 0}, and the actual ranking was {5, 2, 3, 1, 4, 6, 7, 8, 9, 0}, then the score would be 1/23. The steps in the graph show where individual tests jump from1/23 to the optimal solution of 1. By the 200th generation, four out of 12 of the tests achieved the desired ranking. These results might be a feature of the small search space. In the future, larger data sets should be studied.

[2] Negative weights were not allowed due to software limitations.

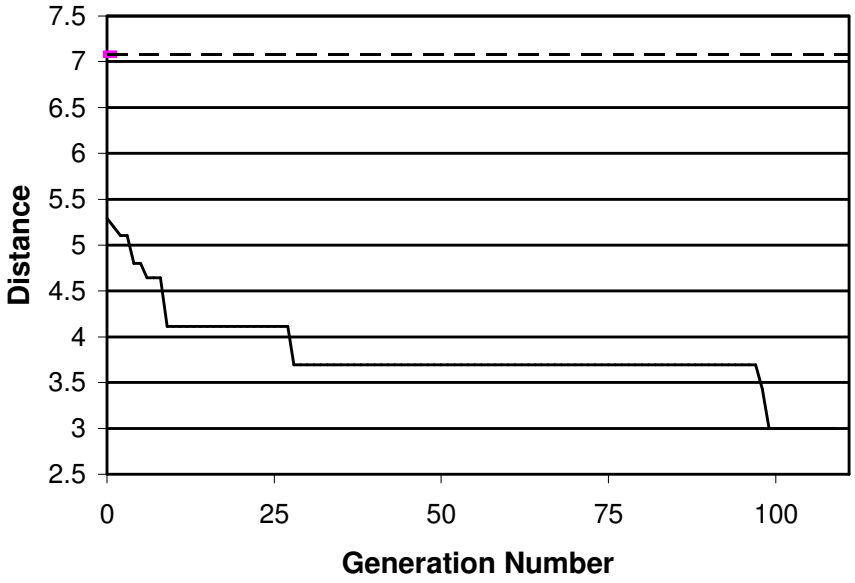


Figure 5: The actual distance between real and desired results plotted over the number of generations. The dashed line represents distance between the rankings of the default weights and the desired results.

The dashed line in Figure 6 represents the performance using the default weights. As mentioned briefly above, the default weights were chosen based on a combination of intuition and trial and error. Interestingly, using the default weights was not an optimal solution for any of the 12 tests, and the distance from the desired ranking in the one realistic test was 8. While these default weights may make sense when considering which HTML and Meta tags are important, it is apparent that better configurations are often possible.

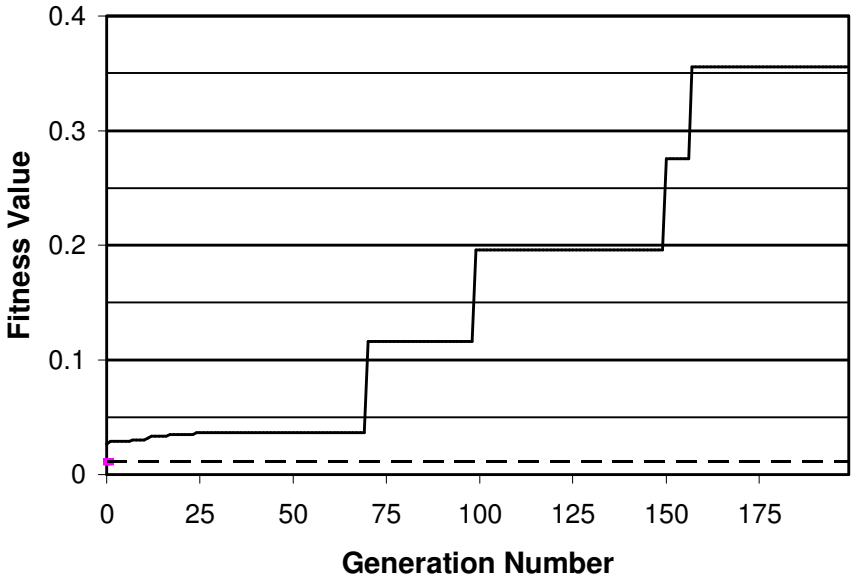


Figure 6: The average fitness values from 12 test cases plotted over the number of generations. The dashed line represents the fitness of the default weights.

One possible argument to support these lower results of the default weights is to hypothesize that the web pages being tested are not properly tagged. If this is assumed true, using standard search engine optimization techniques could help the pages rank better [17], [25]. Unfortunately that position meets with a lot of resistance from real life Webmasters. As discussed in the introduction, improving local search results automatically is one main goal of this research. Therefore, since these test Web pages are representative ones, they are relevant and valid for this research despite apparent shortcomings in design and content.

The fitness function is not a smooth one (consider the -10 bonus for a perfect match and the +100 penalty for a desired page not being listed in the top ten). While

Figure 5 shows the average solution improving over time, it is difficult to see optimization occurring because the fitness values change non-linearly.

In Figure 5, the average actual distance is plotted against the number of generations using a simplified fitness function. The new fitness function does not reward perfect matches, nor does it add a penalty for desired results that do not appear in the top ten actual results.

Using the simplified fitness function, improvement still occurs. However, only one test now achieved the optimal solution. To allow for a better comparison between the original fitness function and the simplified one, consider that the final average distance in Figure 6 (approximated from the final average fitness value) is approximately 2. Comparing that to the final average distance of approximately 3 in Figure 5, it is evident that the simpler function is not able to match the performance of the original equation. However, even the simpler fitness function outperforms the default weights, as depicted by the dashed line in Figure 5.

Structured Knowledge Experiment

For both the LD fitness function and the nDCG fitness function we performed 10 trials. For all trials, the GA using the nDCG fitness function outperformed the GA using the LD fitness function. In fact the LD GA was unable to find a result better than the default weights. As we mentioned earlier, the LD is binary in penalizing improperly ranked results which necessitated the addition of the rank distance from Equation 2.

Examination of the raw output from the GA shows that the rank distance component is not large enough in magnitude relative to the LD. Although the rank distance serves to smooth the fitness landscape, it did not affect the outcome of roulette wheel selection in most cases.

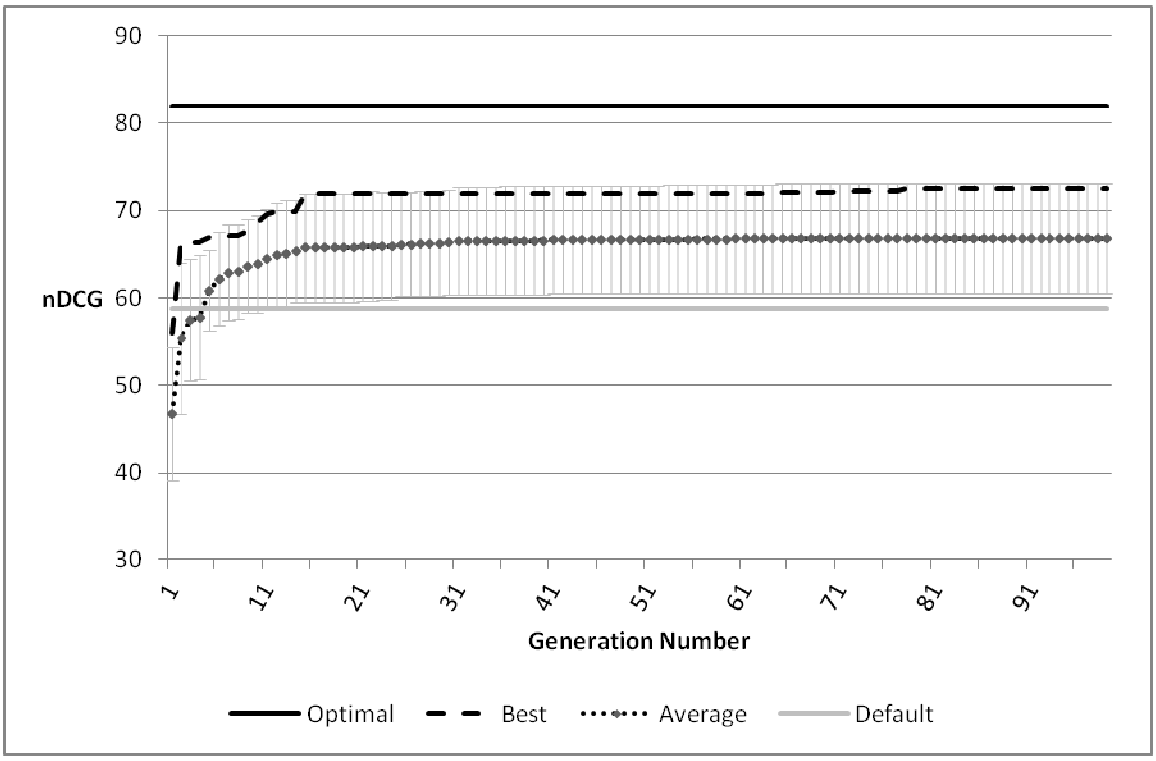


Figure 7: Normalized Discounted Cumulative Gain plotted over generation number. The solid black line is the best solution possible (nDCG = 1 for all search stems). The solid gray line is the nDCG of the default search weights. The dotted line is the average best individual at each generation. The dashed line represents the single best individual fitness at the given generation. The bars above and below the dotted line are the standard deviation of the best individual from each generation.

Figure 7 shows the performance of the nDCG GA. The dotted line represents the average best individual from all 10 trials at a given generation. The dashed line is the best individual from all trials at a given generation. The default weight's nDCG was 58.799 (represented by the solid gray line in Figure 7). Recall that the best possible nDCG is 82 because we are summing the nDCG from 82 search queries.

Table 6 below shows the pair-wise correlation between the 8 search weights as well as the nDCG. Note the strong negative correlation for the 8th search weight, usage boost, and nDCG. Recall from the Background section that the usage boost is not actually a weight applied to a section of the structured content like the other 7 weights. Instead the usage boost is a multiplier applied to the final score based on the popularity of the Answer. For that reason there is a more direct relationship between the search boost and nDCG. Figure 8 is the nDCG plotted over the Usage Boost Weight.

	1	2	3	4	5	6	7	8	nDCG
1	1.000	0.193	-0.051	-0.065	0.006	0.062	-0.076	0.164	-0.168
2	0.193	1.000	-0.066	0.115	-0.159	-0.173	-0.078	0.215	-0.369
3	-0.051	-0.066	1.000	0.022	-0.015	0.009	-0.136	0.051	0.000
4	-0.065	0.115	0.022	1.000	0.001	0.096	0.141	-0.007	-0.099
5	0.006	-0.159	-0.015	0.001	1.000	0.303	0.082	-0.260	0.320
6	0.062	-0.173	0.009	0.096	0.303	1.000	0.188	-0.397	0.366
7	-0.076	-0.078	-0.136	0.141	0.082	0.188	1.000	-0.119	0.142
8	0.164	0.215	0.051	-0.007	-0.260	-0.397	-0.119	1.000	-0.598
nDCG	-0.168	-0.369	0.000	-0.099	0.320	0.366	0.142	-0.598	1.000

Table 6: Pair-wise correlation between the 8 search weights as well as nDCG.

The other seven weights interact in a very complex, nonlinear manner. Figure 9 is the nDCG plotted over Keyword Weight. The complex interaction evidenced by Figure 9 is quite representative of the remaining six weights as well (see Appendix B). The complex interaction is a result of a multi-objective fitness function. There are 82 search stems, each with their own set of relevant documents, which need to be optimized. This results in direct competition between search stems.

For example, although increasing the keyword search weight increases the nDCG of the results for one search stem, it may decrease the nDCG of the results of some other search stem. In other words, if ones search stem's Answers have lots of meaningful content in the keywords section, but another search stem's Answers have irrelevant content in the keywords section, there will be a fitness tradeoff for those two search stems.

Consider that those tradeoffs occur across all seven search weights and between 82 search stems, and the cause of the complexity becomes evident. In contrast, the usage boost weight does not have the complex tradeoff with other search weights, so the correlation to nDCG is more direct.

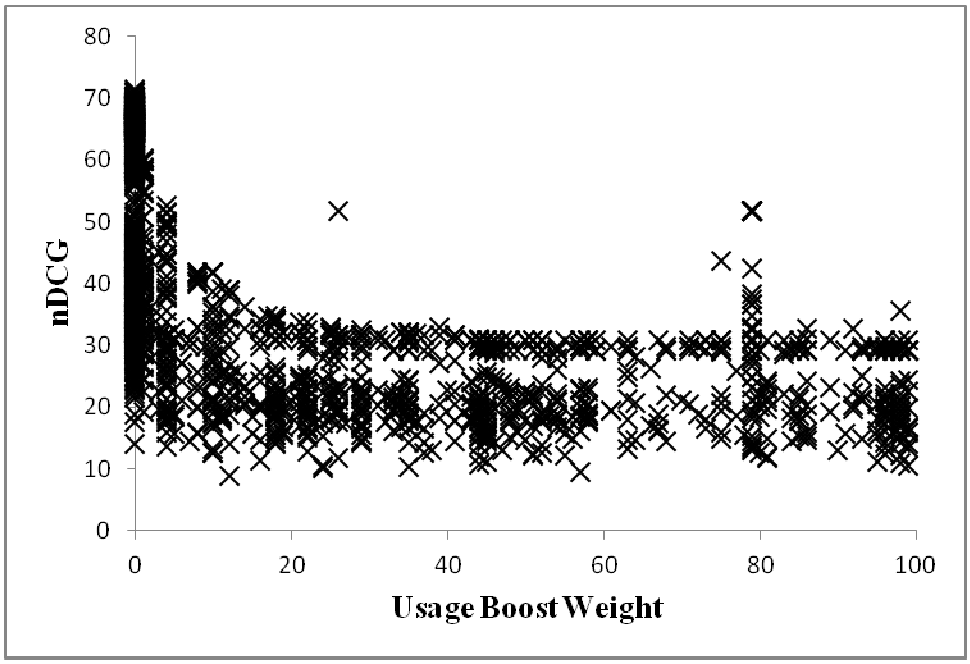


Figure 8: Plot of nDCG over Usage Boost Weight (weight 8) showing negative correlation.

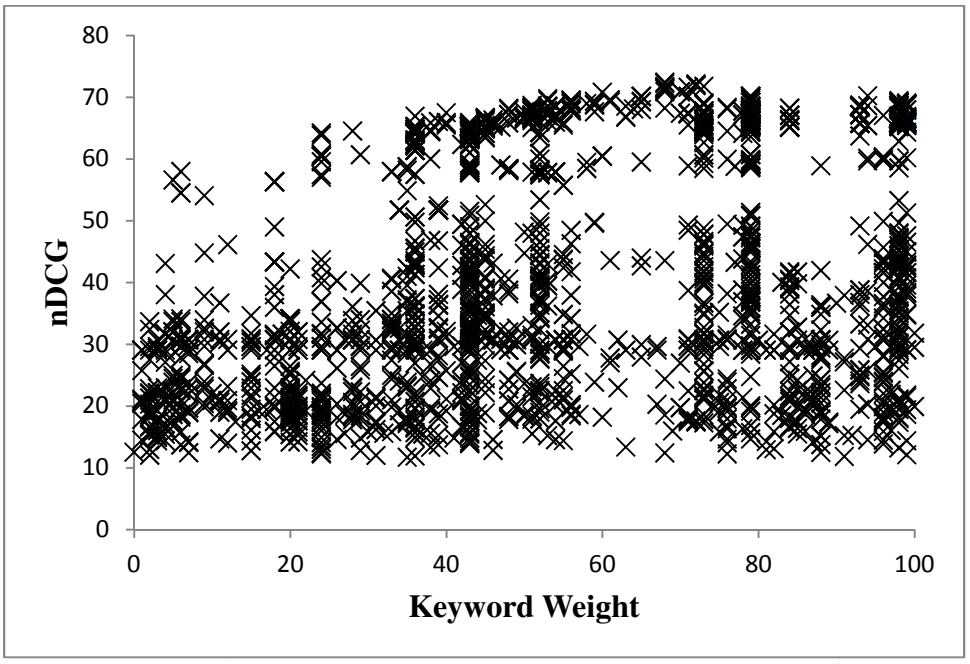


Figure 9: Plot of nDCG over Keyword Weight (weight 5) showing slight positive correlation.

DISCUSSION

Normalized Discounted Cumulative Gain vs. Expected Reciprocal Rank

Chapelle et al. suggest that nDCG assumes that the relevance of a result at a given rank is independent of the results that appear at higher ranks [3]. In other words, the relevance of a document appearing at rank three would be the same regardless of the relevance of the documents appearing at ranks one and two.

They contend that if the documents at ranks one and two are highly relevant, the document at rank three has a reduced probability of being clicked compared to a ranking where the first two documents are low relevance. This is commonly referred to as the Cascade model of presentation bias [4], [5].

According to Chapelle et al., this creates a problem because, given five relevance ratings (Perfect, Excellent, Good, Fair, and Bad), nDCG will grade a ranking set of twenty Good documents higher than a ranking set of one Perfect document and 19 Bad documents. More concisely, their argument hinges on the assumption of Equation 7.

$$\sum_{i=1}^{20} R_{Good} > R_{Perfect} + \sum_{i=1}^{19} R_{Bad}$$

Equation 7: Assumption that total result relevance within a ranked set is not balanced or zero-sum.

Rank	1 Perfect, 9 Bad	10 Good	Actual	Actual Tuned
1	169	17	20	43
2	0	17	16	33
3	0	17	12	23
4	0	17	33	20
5	0	17	23	16
6	0	17	9	12
7	0	17	12	12
8	0	17	1	9
9	0	17	43	1
10	0	16	0	0
Total Raw Relevance	169	169	169	169
nDCG	1	.526777	.542196	.707896

Table 7: Comparison of Raw Relevance and nDCG for a variety of rankings for the search stem "PTA".

However, the relevance measure used in the Structured Content Experiment was zero-sum. Specifically, given *n* instances of a search stem in the clickstreams table, the total relevance that can be assigned to all ranks is n. See Table 7 for several examples based on the search stem "PTA". Column 2 shows hypothetical search results where the first results has perfect relevancy and no other results are clicked – analogous to Chapelle's 1 perfect, 19 bad example. Column 3 shows hypothetical search results where each result has the same relevance. Column 3 is analogous to Chapelle's 20 good search results. The 4th column shows actual search ranks for the search stem "PTA" and the 5th shows the same results after tuning the search weights. The bottom row shows the nDCG for each set of results.

Notice that regardless of how the relevance is distributed, as long as it is a zero-sum system, the ranking independence assumption is not valid and the Cascade model discussed by Chapelle et al. is assumed automatically.

Therefore, we chose to implement the most common nDCG measure instead of the more recent Expected Reciprocal Rank measure that was proposed by Chapelle et al.

Search Recency Boost

Although we chose 8 search weights to tune for the structured content experiment, other weights exist that require tuning. Among them is the search recency boost weight. This weight boosts a search result based on how recently the Answer was last updated. Because this weight's behavior is dependent on action taken by the knowledge manager, rather than the structure of the content or user's search behavior, it was initially excluded from these experiments. However, for completeness we briefly examine this variable below.

First, we need to consider how the Oracle RightNow knowledge base is maintained. Answers are reviewed by a knowledge engineer or the knowledge manager at least every 90 days. Most Answers get reviewed before 90 days due to reuse. Figure 10 shows the distribution of Answers with respect to their age.

Figure 10: Graph showing the actual distribution of Answers with respect to their age.

Note that the behavior seen in Figure 10 shows how irregularly Answers get updated. There is a cyclical nature with a significant number of updates occurring during the week, and very few over the weekends. In contrast, Figure 11 shows a smoother distribution where the Answers' ages are more directly related to the age of the content rather than an artificial review or reuse process.

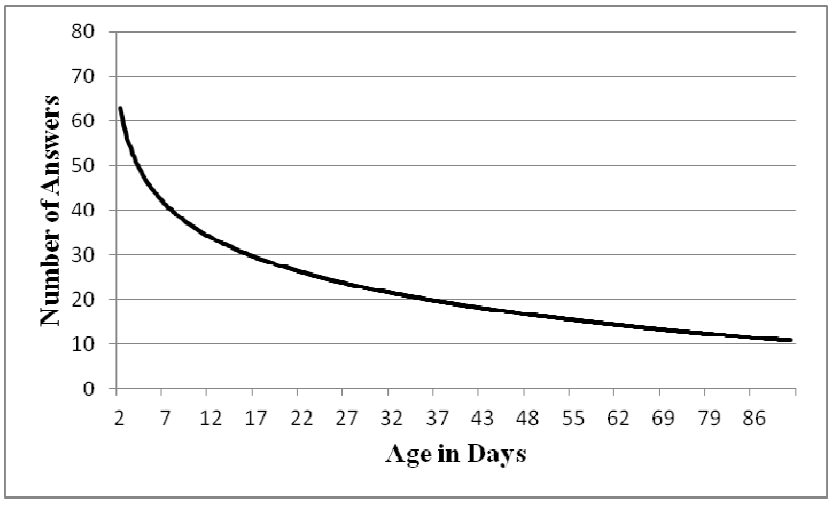


Figure 11: Model of Answer age where Answers' ages are based on reuse and age of the actual content.

Figure 12 shows the performance of the search engine over the range of possible values of the recency boost search weight (holding all other weights constant). We see that the complexity and noise of the recency model shown in Figure 10 renders the recency boost useless. However, in future work, we will experiment with alternate content review procedures in hopes of making better use of this search weight.

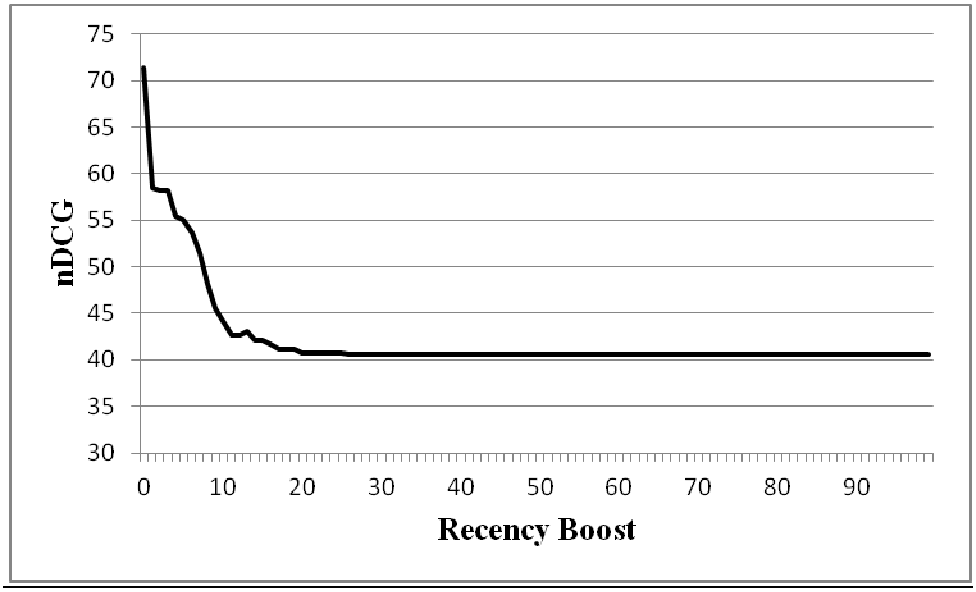


Figure 12: Normalized discounted cumulative gain plotted over the range of possible values of the recency usage boost search weight.

CONCLUSIONS

Feasibility of the GA

A genetic algorithm is not a magic bullet that can configure a search engine to rank pages perfectly. However, as this work shows, the performance of a local search engine can be improved. The level of improvement attainable remains a future research question. Multi-objective genetic algorithms are a possible research path to optimize the search engine weights for larger, more complex page sets [10].

As mentioned in the Results section, the fitness function for the GA used in the web content experiment was not very smooth because of the addition of the perfect result reward of -10 and the missing result penalty of 100. That function performed better than using just the distance and ignoring the two special cases, but it seems likely that a smoother function that includes the special cases would perform even better. In the structured content experiment, the normalized discounted cumulative gain proved to be a much smoother fitness function and did not display the stair-step pattern we saw in the web content experiment.

It is clear that using standard IR relevance measures such as normalized discounted cumulative gain [26] or expected reciprocal rank [3] have several important benefits. First, they create a smooth fitness function which cleanly accounts for positional bias [5]. Second, they allow for meaningful comparison between search technologies throughout the industry, creating a convenient standard for comparing results at

conferences like Text REtrieval Conference (TREC) [2]. Finally, standard IR relevance measures increase consumer and buyer confidence.

It is interesting to note that, although the GA was able to improve on the default search weights in both web content search and structured content search, the default weights performed much better in the structured content search setting. This is due to the additional structure imposed upon Answers. By separating various fields (body, subject, question, etc.), the knowledge manager consciously adds content in a way that closely matches the search engine design. In web content search, those fields are not specified and there is much more variance in the way the content is organized by the web master.

Creating Datasets

Because creating a batch file of desired rankings is tedious and time-consuming work for the Webmaster, it is worthwhile to explore alternate methods of discovering that information. As was mentioned in the approach section, it may be possible to unobtrusively collect information from users about which pages should be ranked higher or lower [22], [1]. When a user conducts a search, statistics can be kept concerning what results the user chooses. The user's final choice could be given special attention since it may be assumed that the user has found the information they were looking for. However, caution is required when using implicit user data because the statistics can be easily misinterpreted. For example, if a search engine returns 10 results per page, it is likely that

the first 10 results will be clicked more often largely because they are on the first page, and not because they are more relevant.

More research is required in the area of implicit user feedback before it can be relied upon for the initial training data. A more reasonable approach might be to bootstrap the search engine with the batch file created by the Webmaster, and then later refine the search engine by cautiously adding the data from implicit user feedback. This will be an interesting and challenging area to study when this research is put into use by a real world search engine and actual implicit feedback can be gathered.

In the structured content experiment we took advantage of the additional structure to assume a more transactional user behavior. On a normal web page, a user leverages the highly connected link structure to browse to appropriate content. Thus, understanding user's intent based on clickstream data is very difficult [15], [16]. Answers are used in a more linear transactional method: users search and then perform some number of Answer views. Answers are generally not linked together so the intent of the search can be more closely coupled with the Answer view.

We found that, although data mined from clickstreams provide a good proxy for manually generated training data, there is still a need for gap analysis. Because clickstreams only provide information on what users will click on if the content exists, it does not provide information on missing content. It might still be possible to automate this analysis. For example, we could present users random search results to establish a model for click behavior over low relevance results. We could then compare the click

pattern of a set of search results to the random model. The closer they match, the more likely that there is an information gap that needs to be filled with additional content.

Thus, although some initial progress has been made, much research remains. We are excited to continue exploring this problem.

RECOMMENDATIONS

Search Tuning Process

Create a Baseline and Monitor

We recommend measuring and tracking a standard IR relevance metric such as nDCG. By recording nDCG for a set of search stems every day, a knowledge manager can get a sense of how the relevance of the search results is trending. After watching this trend for a time, the knowledge manager should be able to tell how often search and content tuning is necessary, allowing him or her to plan periodic maintenance.

In addition, it is important to track the relevance measure before and after making major changes to search technology, web site structure, or content. This will alert the knowledge manager to unexpected negative consequences of the change and allow them to be corrected in a timely manner.

Perform Periodic Search Parameter Tuning

Based on the degradation in performance that is observed by monitoring the relevance baseline, the knowledge manager should periodically tune the search engine parameters. The frequency of this tuning will vary based on the dynamics of the content generation and maintenance, especially if the structure of the content changes.

Although a large scale survey has not been performed, anecdotally it appears that search tuning should take place about once every one to three months.

It is important to remember that each time the search engine is tuned, a new dataset should be generated from the clickstream data.

Perform Periodic Content Tuning

After tuning the search engine parameters, some pages will still not be ranked correctly. The knowledge manager will need to generate a click histogram for each search stem with a low nDCG, similar to Figure 1. The histogram can then be used to visualize which Answers or web pages need to have content tuned.

For pages that are still ranked incorrectly after the search engine is tuned by a GA, search weights can be used to make further recommendations for improvement. For example, if the description weight is large and the misclassified page has no description, it seems appropriate to recommend writing a description for the page.

Another way to deal with a misclassified page is to automatically generate a value to use in one of the search engine specific meta-tags. This allows artificial content to be added to a page without altering its visible contents. In general, altering the search engine specific meta-tags should not affect the page's ranking with other Internet-wide search engines [24]. Therefore, to deal with problem pages, the program could add artificial content to website meta-tags as a temporary solution, and then make recommendations for additional content to be added later by the Webmaster as time allows.

The strategy of adding meta-content can also be applied to structured content by adding search terms to the keywords section. One way to accomplish this would be to mine the clickstreams and add the search term to the Answer's keywords section. This is similar to creating a tag cloud based on the users' searches. There is the issue of noise in

the clickstream. This can be overcome by only adding a keyword after the search stem and Answer pair appear together several times in the clickstream. In our limited testing, this strategy of adding keywords based on frequent pairings of the search stem and Answer has shown to be very successful. If the search engine parameters are tuned again after adding keywords based on the clickstream content, it will overfit to that content. This is because the fitness function and training data are derived from the same clickstream content. Overfitting in this case means that the search engine is optimized for content at a given point in time, but the settings will not generalize for content in the future.

In this domain, overfitting is actually a desirable trait to some extent. It allows the knowledge manager to tune to perfect or nearly perfect search results at a point in time. If the tuning is not periodically performed, the overfitting will cause the relevancy to decay over time. However, as long as the knowledge manager repeats the search tuning whenever the relevance measure drops too low, the overfitting works well. For this reason, we suggest a second round of search parameter tuning after adding the keywords.

The search engine configuration file weights can also be used to make design change suggestions. For example, if the GA finds that the title tag's weight should be very small, a recommendation could be made to use title tags more accurately. (In general, the title of a web page is considered to be a very important aspect by many Internet-wide search engines.) In the results from the web content experiment (Table 1), the relatively small weight of the keywords tag likely indicates that the keywords are written poorly.

Additional Datasets and Testing

In this work, I have shown that using a genetic algorithm to tune search engines is feasible and results in much higher search result relevancy. I look forward to applying this method to additional datasets to further validate the process and results.

The dataset used in this work was created to facilitate business to business (B2B) interactions. This means that the consumers of the content are regular users and somewhat familiar with the content. In contrast, a business to consumer (B2C) organization needs to pay special attention to the organization and presentation of content to facilitate a high volume of infrequent users. It will be interesting to investigate how the type of content and behavior of the users affect the tuning processes I have described.

REFERENCES

[1] Boyan, J., Freitag, D., Joachims, T., A Machine Learning Architecture for Optimizing Web Search Engines. *Proceedings of the AAAI Workshop on Internet Based Information Systems*, 1996.

[2] Boytsov, L., & Belova, A., Evaluating Learning-to-Rank Methods in the Web Track Adhoc Task. *TREC-20*, November 2011, Gaithersburg, Maryland, USA

[3] Chapelle, O., Metzler, D., Zhang, Y., & Grinspan, P., Expected Reciprocal Rank for Graded Relevance. *CIKMACM* (2009) , p. 621-630.

[4] Chapelle, O., & Zhang, Y., A Dynamic Bayesian Network Click Model for Web Search Ranking. *Proceedings of the 18th International Conference on World Wide Web. New York, NY: ACM,* 2009.

[5] Craswell, N., Zoeter, O., Taylor, M., & Ramsey, B., An Experimental Comparison of Click Position-Bias Models. *Proceedings of the international conference on Web search and web data mining* New York, NY, USA: ACM (2008) , p. 87--94.

[6] Daida, J., Ross, S., McClain, J., Ampy, D., & Holczer, M. Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming. *Genetic Programming 97. San Francisco, CA: Morgan Kaufmann Publishers*, 64-69, 1997.

[7] Daida, J. M., Ampy, D. S., Ratanasavetavadhana, M., Li, H., & Chaudhri, O. A.. Challenges with Verication, Repeatability, and Meaningful Comparison in Genetic Programming: Gibson's Magic. *Proceedings of the Genetic and*

Evolutionary Computation Conference. San Francisco, CA: Morgan Kaufmann Publishers, 1851-1858 (Volume 2), 1999.

[8] Durbin, S., Warner, D., Richter, N. & Gedeon, Z. Management for Web-Based Customer Service. *Organizational Data Mining: Leveraging Enterprise Data Resources for Optimal Performance. Edited by Nemati and Barko. Idea Group Inc*., 92-108, 2004.

[9] Durbin, S., Warner, D., Richter, N. & Gedeon, Z. Information Self-Service with a Knowledge Base that Learns. *AI Magazine, 23*(4), 41-49, Winter 2002.

[10] Fonseca, C.M., Fleming, P.J., Genetic Algorithms For Multi-Objective Optimization: Formulation, Discussion And Generalization. *Genetic algorithms: Proceedings of the Fifth International Conference, Morgan Kaufmann, San Mateo, CA*, 141-153, 1993.

[11] GALIB, A C++ Library of Genetic Algorithm Components, http://lancet.mit.edu/ga

[12] Goldberg, D. E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[13] Hettich, S. and Bay, S. D. The UCI KDD Archive. University of California - Irvine, Department of Information and Computer Science. http://kdd.ics.uci.edu

[14] Järvelin, K., & Kekäläinen, J. IR Evaluation Methods for Retrieving Highly Relevant Documents, *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY: ACM, pp. 41–48.

[15] Joachims, T., Granka, L., Pan, B., Hembrooke, H., & Gay, G. Accurately Interpreting Clickthrough Data as Implicit Feedback, *Proceedings of the Conference on Research and Development in Information Retrieval (SIGIR)*, 2005.

[16] Kelly, D. & Teevan, J., Implicit Feedback For Inferring User Preference: A Bibliography. *SIGIR Forum*, Vol. 37, No. 2. (September 2003), pp. 18-28

[17] Kent, P., *Search Engine Optimization for Dummies*, Wiley, 2004.

[18] Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8), pp. 707–710, February 1966.

[19] Meysenburg, M., & Foster, J. Random Generator Quality and GP Performance. *Proceedings of the Genetic and Evolutionary Computation Conference. San Francisco, CA: Morgan Kaufmann Publishers*, 1121-1126 (Volume 2), 1999.

[20] Mitchell, M., An Introduction To Genetic Algorithms. MIT Press Cambridge, MA, USA 1996

[21] Nonaka, Ikujiro (1991). The knowledge creating company. *Harvard Business Review* 69 (6 Nov–Dec): 96–104.

[22] Qi, H., Hartono, P., Suzuki, K., Hashimoto, S., Sound Database Retrieved By Sound*, Acoustical Science and Technology, Vol. 23*, No. 6, pp. 293-300, 2002.

[23] RightNow Technologies, On Demand Customer Relationship Management Software, http://www.rightnow.com

[24] Sullivan, D., *How To Use HTML Meta Tags, Search Engine Watch*. http://www.searchenginewatch.com, December 5, 2002.

[25] Thurow, S., *Search Engine Visibility*, New Riders, 2003.

[26] Weimer, M., Karatzoglou, A., Le, Q., & Smola, A., CoFi$^{\text{Rank}}$ Maximum Margin Matrix Factorization for Collaborative Ranking. *Advances in Neural Information Processing Systems 20* MIT Press (2007) , p. 1600, 1593.

APPENDICES

APPENDIX A:
Sample Knowledge Base Answer

Best practices for using RightNow Service

Answer ID 1488 | Last Review Date 03/28/2012

What are best practices for implementing and using our RightNow Service application.

RightNow Technologies has developed Service Best Practices from our growing customer base. These best practices are designed to help companies maximize RightNow Service site effectiveness.
During your implementation, you and your Implementation Project Manager will evaluate your site. Use the links below to view specific items and descriptions for each of the major tasks listed below.

Note: The links below do not open in a new browser window. If you wish to return to this page after clicking a link below, click the Back button to return to this answer.

Process-Related Best Practices

- **Phase 1: Initial Release**

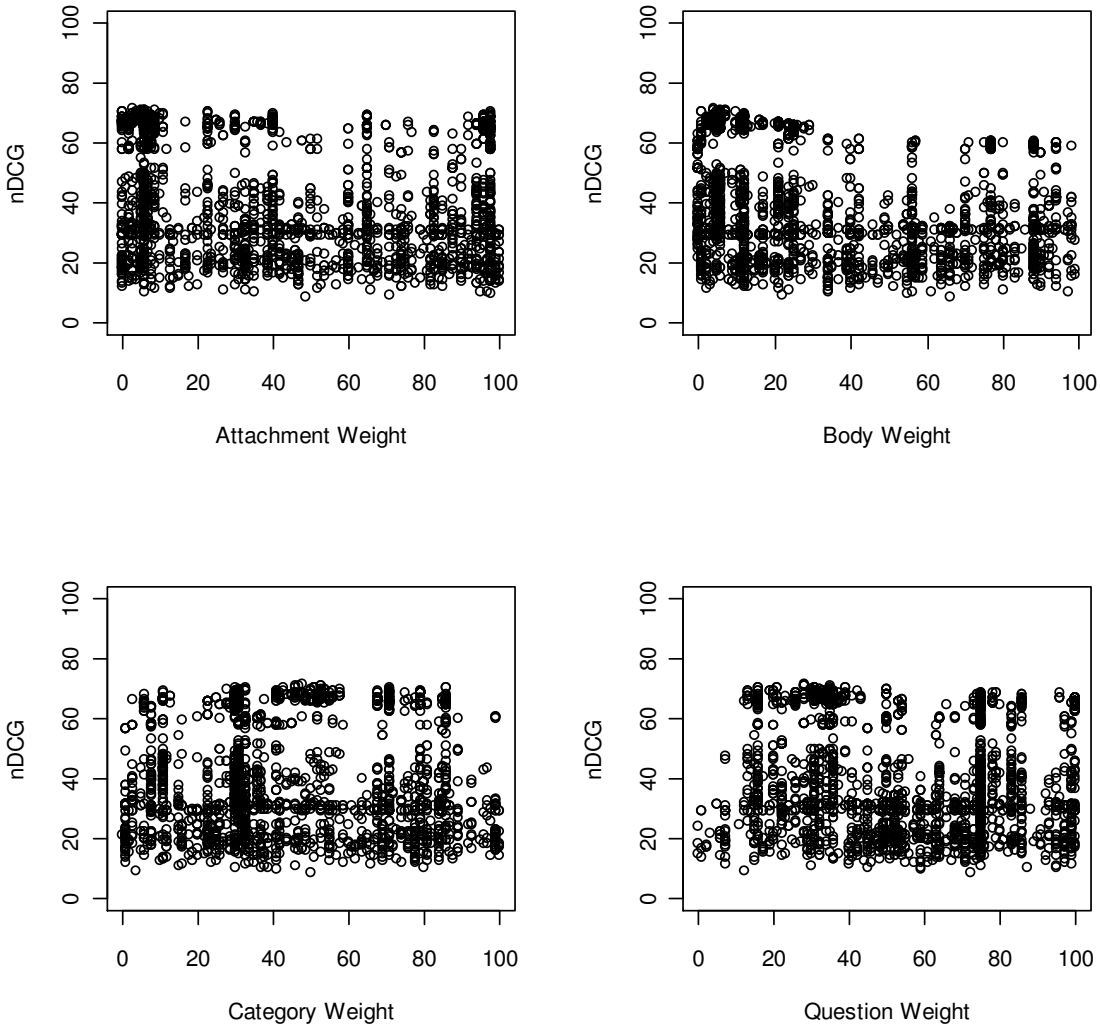
  Project Team
  Project Plan
  Software on Current Major Release
  Funnel Through Self-Help
  Centralize Inbound Requests
  Business Rules
  Standardize Site Design
  Staff Efficiency Features

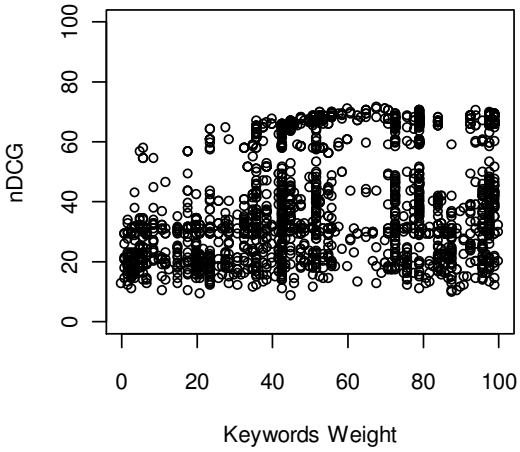
- **Phase 2: Post-Launch**

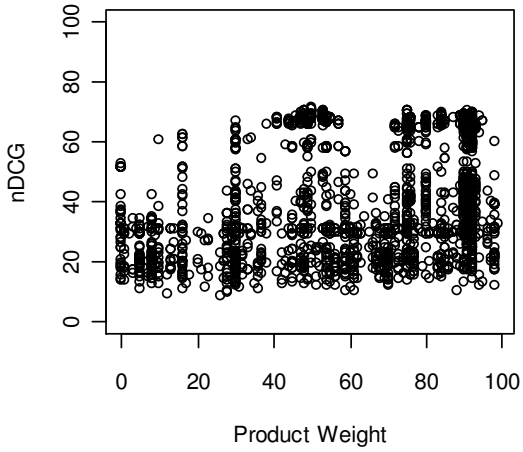
  Knowledge Base Management
  Customer Satisfaction
  Promote eService

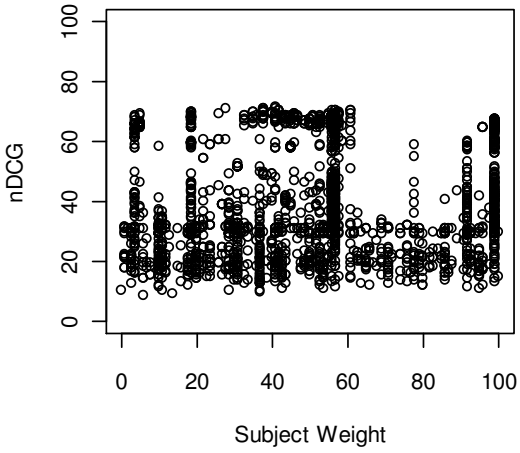
In addition, the Best Practices Implementation Guide is designed to help companies successfully deploy RightNow Service. This guide summarizes recommended best practices gleaned from over thousands of RightNow customer Implementations and Tune-Ups of both large and small companies. Use these best practice guidelines and your project plan to assist in a successful implementation so that your company can begin to accomplish your goals.

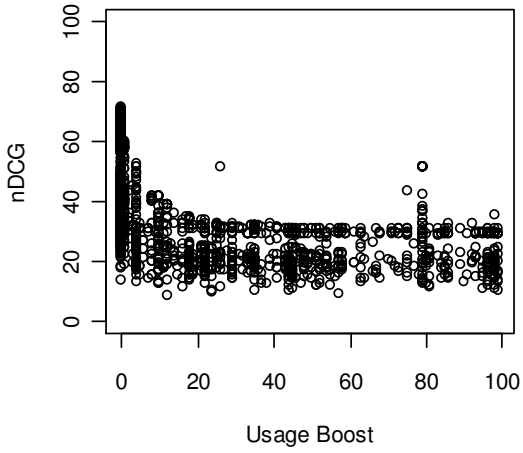
APPENDIX B:
Plots of nDCG vs. Structured Content Search Weights




nDCG
0
20
40
60
80
100
0
20
40
60
80
100
Keywords Weight


nDCG
0
20
40
60
80
100
0
20
40
60
80
100
Product Weight


nDCG
0
20
40
60
80
100
0
20
40
60
80
100
Subject Weight


nDCG
0
20
40
60
80
100
0
20
40
60
80
100
Usage Boost