

Home Page Finder

Omid Fatemieh, Kashif Manzoor, Arpit Jain, Aditya Ramani

December 12, 2005

Abstract

Over the last few years, the information explosion has resulted in a unique problem, it has become increasingly difficult to locate what you want. Such a problem has been solved to some extent by commercial search engines. However, it might be necessary to develop applications with enhanced specificity over particular domain. In our case, we have addressed the problem of locating homepages or conferences, given a suitable query. We develop an additional processing layer over the existing search infrastructure, to solve this problem. Our results turned out to be quite satisfactory.

1 Introduction and Previous Work

Developing applications that address specific search niches is not a new problem. Such applications need to find a balance between developing generally applicable techniques and specific domain specific optimizations. There are 2 major approaches to this problem ([1],[2]) :

- Indigenous research that tends to solve this problem through exhaustively crawling the internet and archiving the crawled web page, and then sniffing out the personal homepages out of these crawled results. e.g. DBLife
- Using the existing web page and other information retrieval systems and applying techniques on these results to sniff out the personal web pages. e.g. Home Page Finder

Whereas the early and mid nineties mostly saw research being done in the first category (all the commercial search engines like google, msn, yahoo are the result of this research); the later half of nineties to date has seen research conducted mainly in the second category.

A lot of systems have exploited this idea and have developed increasingly complicated and niche search utilities. Ahoy [1] was the first such system that used three external systems and applied simply yet extremely successful heuristics to retrieve the users homepage. Ahoy used three external systems for posting its queries. One was a traditional MetaCrawler (similar to commercial search engines like google), other was an email directory service and the third was an internal persons institution name provider service. It then applied heuristics to mine the correct homepage out of all the possible web pages provided by the MetaCrawler using the output of the e-mail directory service and the internal server to help it make the right decision. Ahoy showed high accuracy and outperformed all the commercially available search engines. However, by using three external systems Ahoy increased its dependence on the information provided to it as input. Additionally Ahoy's much touted auto-url generator which could even try to find homepages that are not even present in the MetaCrawlers repository only worked on the educational domain. Ahoy was presented in 1997, and the search technology since then has advanced a lot to an extent that a lot of things that Ahoy was performing are now embedded into the commercial search engines.

McCallum and his group [2] takes the idea of Ahoy one step further and instead of finding the homepage they attempt to find a whole social network to which the user belongs - based on the personal information that they mine from his email and the web. This is an interesting area of research; however it uses the users emails as the starting point which is quite different than what we have as the starting point. Given only the user name [2] can not be expected to perform at all. Our system, although only solves a small subset of what [2] attempts to do, but we are working with much tighter limitations than [2]. McCallum and his group did follow up work in [2] where they attempt to use the persons social network information in disambiguating the users on the web (i.e. given a John their system will bring only that John to which the user is related the most). In doing so they find the users homepage as one of the ways to detect the users social network and in determining that John which relates to the user. Their major focus still remains on using the users email as the starting point. Both [2] and [3] have a community focus in that they operate within the boundaries of a community/social network. We on the other hand have not restricted ourselves to only a particular social community. We attempt to find the homepage of any person irrespective of his social network. In this way our approach and problem statement is more close to [1]. But [1]'s approach has been outmoded with time, and we have rejuvenated that approach by making it much simpler and less dependent on external systems.

Thus like some of the previous work on this problem, in our project, we would like to leverage the excellent ranking provided by commercial search engines for our search. We essentially are trying to refine the work done by existing search engines. This ensures that, in the worst case our methods would perform about the same as the search engine, working alone.

2 Architecture of Homepage Finder

The architecture of Homepage Finder is shown in 1. It has 4 basic stages : the Google query , query result consolidation, heuristic application and heuristic weighting and consolidation. Homepage Finder uses Google as its base search engine. Google API [5] to pose various queries to Google and analyze the top search results generated by Google. The main component of our system is thus Google API. It provides us with some candidate answers which we use to figure out the homepage of the researcher or

a conference. It makes the reasonable assumption that the homepage of the researcher or the conference is somewhere in the search results generated by the Google. We ensure that by cleverly rewording the user query. We try to keep the list of our candidate urls as short as possible, without losing out on correctness.

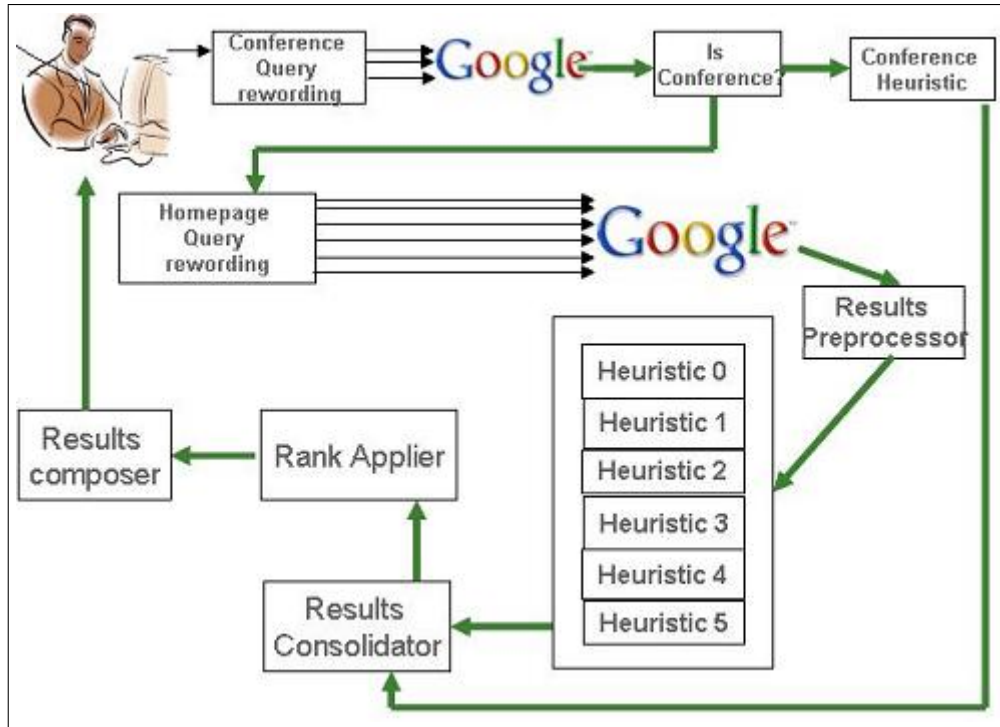


Figure 1: Architecture of Home Page Finder

The user query is either for the homepage of a researcher or a conference. We will now explain how we process the query.

- Our first component determines whether the keyword query posed to Homepage Finder is a researcher or a conference name. This is determined by posing three search queries to the Google API. We analyze the top 10 search results of each query provided by the Google API and using some heuristics figure out whether the user is looking for a conference or a researcher. The reason we distinguish between these two is we have different set of heuristics for each of them. If the user is looking for a conference homepage then we apply conference heuristics else if it is a researcher then we apply researcher heuristics.
- In the second step, we further do query rewording to determine the homepage. For the case of homepage of a conference, we pose one search query to the Google API and for the case of homepage of a researcher, we pose six search queries to the Google API. The results from these queries go to the result consolidator component. Result consolidator combines the result from all the queries and apply the corresponding heuristics. We do some preprocessing of the urls like removing duplicates and generate the candidate list of urls from which we need to find the homepage.
- Rank consolidator sends its list of candidate urls to Rank Applier. Here, heuristics are applied to each candidate url in the list and is assigned a score. We put different weights to heuristics for calculating the final score for the candidate url. The heuristics weights are assigned depending upon their importance. These weights were determined by running some test cases and analyzing their performance. Rank Applier sorts the candidate urls in the increasing order of their scores.
- Finally, the ranked results goes to Result Composer. For each url, we also show the corresponding snippet. This is the same as shown by Google. Result composer puts the candidate url with the highest score at the top. We compare our url ranking with how Google ranks these candidate urls

when we pose the user query to it. Ideally, when we query Google for a name of a researcher, the homepage should be the first result. However, this is not the case for a number of cases. For all the test cases, our system always ranks the homepage of the researcher or the conference as the first result. This clearly shows that Homepage Finder performs better than Google.

In the following sections, we will explain each of the components in detail. We then go on to test our methods against the results of the major commercial search engines (Yahoo, MSN and Google), and then describe how our methods could be generalized.

3 Google API

The Google Web APIs service is a beta web program that enables developers to easily find and manipulate information on the web. Developers can issue search requests to Google's index of billions of web pages and receive results as structured data, access information in the Google cache, and check the spelling of words. Google Web APIs support the same search syntax as the Google.com site. We thought it wise to use it as we can make use of the relatively high quality of the results provided by the Google search engine.

Google provides each developer who registers to use the Google Web APIs service a limit of 1,000 queries per day. Google supports the use of several special query terms that allow the user or search administrator to access additional capabilities of the Google search engine. Specifically, we have used the following search capabilities:

- Title Search (term) - If you prepend "intitle:" to a query term, Google search restricts the results to documents containing that word in the title. Note there can be no space between the "intitle:" and the following word.
- Title Search (all) - Starting a query with the term "allintitle:" restricts the results to those with all of the query words in the title.

Note: Putting "intitle:" in front of every word in your query is equivalent to putting "allintitle:" at the front of your query.

Search requests submit a query string and a set of parameters to the Google Web APIs service and receive in return a set of search results. Search results are derived from Google's index of billions of web pages. Each time a search request is issued to the Google service, a response is returned back. Search result consist of a bunch of webpages with their title, url and snippet. Google API has built-in functions which can extract title, url, snippet from a webpage returned as part of a search request.

We use to Google API to pose a number of reworded queries. This rewording is domain specific, and we are trying to exploit information about the domain that is not likely to be provided directly by the user. Our method has about 6 different types of rewordings, where we attach other words that are likely to occur on a researcher homepage and not elsewhere. We have focused on the specific problem on finding researchers though it could also be generalized. Sample words are 'homepage', 'resume', 'publications'. These are appended to the existing query string. The results from these form the base set of candidate urls, on which we apply finer-grained heuristics and try to identify the top candidates. The heuristics that we apply are described in the next section.

4 Heuristics

In this section we cover the heuristics that we used to boost the desired pages up in our final ranking of the pages. The heuristics we used are based on a number of observations that are often (not always) true for many of personal or conferences homepages. We admit that these heuristics are to an extent *ad-hoc*, but we do maintain that domain specific details always tend to be so. It would be an interesting problem to come up with heuristics that were both relatively broadbased yet works well on the domain. Also, we suspect that the search engine would have already covered heuristics such as that. These are our observations for a personal homepage (throughout this chapter we use the query Anhai Doan as a homepage query and SIGMOD as a conference query):

- It contains the persons name in title.

- It contains some variation of the persons name in the URL, e.g. anhai, or adoan.
- The stem of several other related pages is the same as this page
- It contains the words home, page, site, web, web site in its contents.
- It contains the persons contact information, publications, pictures, profile, resume etc.
- It Has the words home, page, site, web, web site in its contents.
- The URL ends with a / or index.html, default.htm.

However, we should mention that there are always exceptions to these rules. For example some of the personnel directory websites may violate some of the properties above. Also, many of the dynamic web sites may take in a persons name as a GET parameter. On the other hand, some appearances of a persons name on popular web sites may have some of the properties above despite the fact that they are not a web site (e.g. Amazon reviews, or researcher profile page at DBLP). However, as we will see in our results, by proper use of a combination of the above mentioned observations, we could get promising results, which suggests that they hold in many cases. The key to our work is to be able to deal effectively with these hits that come up high simply because of the base website, i.e. Amazon.

4.1 Conference or Homemage?

Our first set of heuristics include the ones that we use to determine if the posted query is a person name of a conference name. First suppose that the given query is a conference name. Now consider the following queries:

Q1 Raw Query (e.g. SIGMOD)

Q2 Intitle query (e.g. intitle: SIGMOD conference OR symposium)

Q3 Allintitle Query(e.g. allintitle:SIGMOD conference OR symposium)

Now the question that Does the keyword represent a conference? can be answered by considering the following observation. If denote the number of results returned for a query Q as $num(Q)$, in a normal case, $num(Q3)$ must be zero for a person. Also the $\frac{num(Q1)}{num(Q2)}$ ratio should be higher than a threshold. We set this threshold to 0.4 based on our experimental tests. We used this simple heuristic to decide if the posted query is a conference name or a person name. Then if we decide it is a conference, we simply return googles first result, which almost in all cases proves to be working pretty well.

4.2 Homepage Heuristics

The heuristics that we use to find a persons homepage are mostly based on the observations that were mentioned in the beginning of 4. They can be divided into two groups; the heuristics in the first group mainly look at the content, as opposed to the second set of heuristics which are mainly focused on the returned URLs.

4.2.1 Content-Based Heuristics

In order to prevent from re-inventing the wheel, we look at the content by submitting different queries to google. By this outsourcing, not only we avoid re-inventing the wheel, but also use the cutting edge search technology with considerable confidence. We submit a number of reworded queries to google as follows. The relative weight specifies the weight that we used to boost the results returned from each of the queries for our final ranking. For each of the queries below, we only look at googles top ten results, but we dont look at the rank that google provides for them. To avoid from being too influenced by googles ranking system, we just treat all the ten results returned for each query the same.

Q1 Raw Query - Anhai Doan [relative weight=1]

Q2 Intitle query - intitle:Anhai Doan [relative weight=3]

Q3 Allintitle query - allintitle:Anhai doan home OR homepage OR webpage OR "web site" OR personal OR page [relative weight=2]

Q4 Resume/CV - Chengxiang Zhai resume OR "CV" OR "Curriculum Vitae [relative weight=0]

Q5 Chengxiang Zhai publications [relative weight=0]

Q6 Chengxiang Zhai contact [relative weight=0]

The results from the last three queries are assigned relative weight of zero. The reason for this is that we only use them for testing and boosting the results returned from the first three queries and we don't use them directly as our final results. More details on the boosting procedure are available in the heuristic weighting and results sections.

4.2.2 URL-Based Heuristics

We use a number of heuristics for this purpose. First we use a URL preprocessor which assigns a very low score to the URLs that contain the ? character. The reason for this is that we want to avoid giving a high rank to CGI URLs that are usually not homepages.

Our second heuristic looked for variations of the user's name in the URL. As an example, for Anhui Doan, such variations can be anhui, doan, adoan, doana, etc. We gave a relative weight of two to this heuristic. Our third heuristic simply looks at words like home, homes, homepage, people, etc. An example of this is one of the author's homepage at UIUC (www.cs.uiuc.edu/homes/sfatemi2/). We gave a relative weight of one to this heuristic.

The fourth heuristic that we present in this section tries to match the last characters in the URL with /, index.*, default.*. Again, based on our experimental pre-evaluations, we decided to give a relative weight of one to this heuristic.

We also additionally noticed that often, if the hit was for a website of a person, there were a number of pages that turned up in the list of candidate URLs, and all these had the same underlying stem. So, our algorithm scores each candidate URL for the extent to which it matches with the other candidate URLs. The higher the score, the more likely we would think for it to be a website. Of course, one has to understand that most of these techniques are approximate, and there always will be cases when they do not turn in the correct answer.

5 Heuristic Weighting and Consolidation

The act of having multiple heuristics is called bagging. In which we tend to apply various orthogonal techniques (the mutually exclusive heuristics) to reach a verdict. However bagging alone can not guarantee a swift and correct solution. Using multiple heuristics is like having several advisors. On one hand it can improve the chances of making the right decision but on the other if the advisors provide conflicting advice then we need a way to decide who to listen and who to ignore. This is achieved by assigning a weight to the outcome of each heuristic a process called boosting. There are many ways to apply boosting one of the most popular boosting algorithm is Adaboost [6]. This was presented by Freund and Schapire in 1995. This algorithm dynamically adjusts the weights of each heuristic based on a training set until the desired accuracy is reached. It passes the same training set through the heuristic one by one penalizing a heuristic every time it inaccurately identifies a training object. Although our basic boosting approach is inspired by Adaboost but instead of mathematically extensive weight adjustment, as done in Adaboost, we have taken a manual approach in adjusting the weights (i.e. we manually decided how much a heuristic should be rewarded or penalized depending on its outcome).

The results of the heuristics are given in the following section. As is clear, in addition to correctly identifying the homepages several heuristics also identify false positives. Based on these findings the weights were adjusted to penalize the heuristics that produce high percentage of false positives. A more detailed explanation is available in the next section.

6 Results

6.1 Quality of Results

We manually ran several queries on three commercial search engines. We selected the keywords according to the following distribution: 50% persons being famous or semi-famous (e.g. researchers, actors etc) and 50% persons being not so famous (e.g. personal friends, graduate and undergraduate students). A result is considered correct if the homepage of the person is the first one result of the search engine. We calculate error by considering the ranked results as ordinal data where each homepage result can have a

rank: R_{if} is a member of $\{1, 2 \dots M_f\}$ and then applying the Z-Score normalization. (i.e. $Z_{if} = \frac{R_{if} - 1}{M_f - 1}$ where R_{if} is the rank assigned by the search engine i to the correct homepage and M_f is the worst possible rank (considered as 10 in our case). The mean error is calculated over all the test data to give the final overall percentage age error of a search engine. The results are presented in the Table 2 below. As it can be seen in the table, HPF outperforms all other search engines. Although HPF uses google

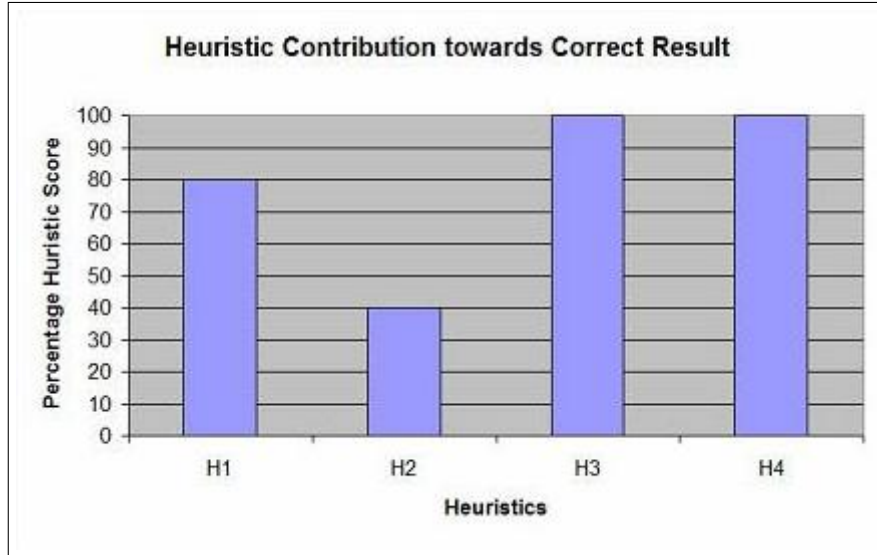


Figure 2: Heuristic contribution to correct result

as its first step but it pays no attention to google's rank, hence its better performance than google is a credit to its own heuristics. Google and Yahoo usually have the same home page in the top 10 results, but if it is not the first result we report this as an error.

6.2 Heuristic Effectiveness

We also performed some tests to evaluate the effectiveness of our heuristics. We performed two sets of evaluation on the heuristics. First, we compared the contribution of each heuristic towards correct results, and next we evaluated the contribution of each heuristic towards false positives. You can see the results from these heuristics in the tables 2 and 3 below. The four heuristics we evaluated were H1: user name in URL heuristic, H2: specific words in URL heuristic, H3: URL pattern heuristic, H4: Stem matching heuristic.

We can see that all heuristics have been contributing to our results significantly. It is really interesting to observe that H3 and H4 had contributed to our results almost all the times, which was far beyond our expectation. We were also really impressed by the results from H1, which has a higher weight and can be a significant indicator of a personal homepage.

The next diagram shows the contribution of heuristics to false positives. The most striking result is H4, which is almost always contributing to false positives. We used these results to set our weighting factors (please see Section 3).

6.3 Limitations in Experimental Results

We encountered a number of limitations in evaluating our system. First of all, our evaluation was not so exhaustive due to manual tagging required and also the limitations of the google's free API. We couldn't find any homepage portal web site to be used as test data.

One of other limitations in our evaluation was that Adaboost Algorithm (see the related work section) was not systematically applied to the heuristic results, instead the weights were adjusted manually based on the error following the general guidelines of the algorithm. Heuristic contributions also, could have

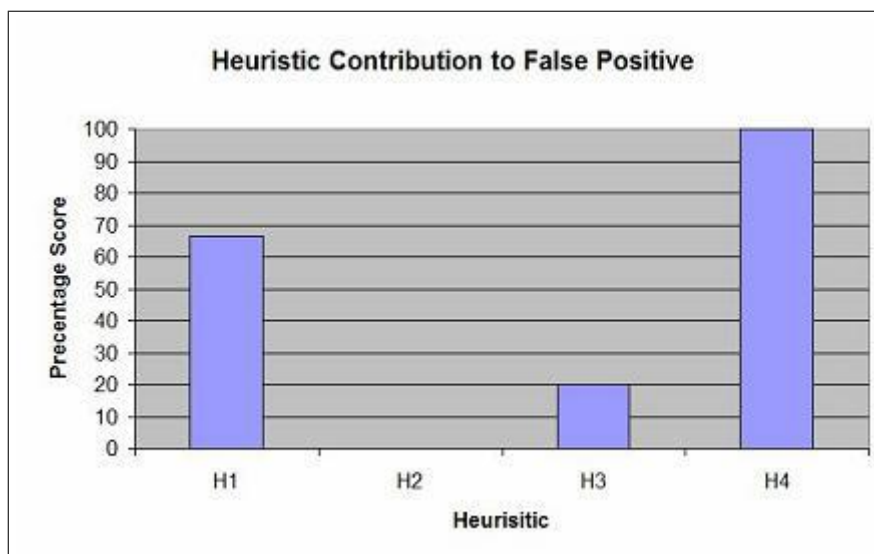


Figure 3: Heuristic contribution towards false positive.

been further investigated similar to how Doan et al. [4] compares the learners. We leave the incorporation of the last two features as our future work for a publication submission.

7 Conclusion and Future Work

Our experience with the problem has made us feel that it is very difficult to improve the efficiency of the search procedure without taking into account the details of the exact domain that we are trying to address. In the future, the key would be to develop systems that have the right combination of general methodologies and domain specific optimizations. Also, it would be interesting to develop a language to model heuristics, and given a specific domain, try to learn the set of heuristics that work well on it. This idea is similar to work done by Doan et. al. [4]. For a large search engine such as Google, even if 1% of the total queries could be contributed to experimental heuristics testing, I think there is a lot of potential. In our opinion, the main drawback in systems developed by us, is not the system *per se* but the issues with being able to test it efficiently.

References

- [1] J. Shakes, M. Langheinrich, and O. Etzioni. Dynamic reference sifting: A case study in the homepage domain. In Proceedings of the 6th World Wide Web Conference, 1997.
- [2] A. Culotta, R. Bekkerman, and A. McCallum. Extracting social networks and contact information from email and the web. In Proceedings of CEAS-1, 2004.
- [3] Ron Bekkerman, Andrew McCallum. Disambiguating Web Appearances of People in a Social Network. In Proceedings of World Wide Web Conference, 2005.
- [4] A. Doan, P. Domingos, and A. Halevy, Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach, . Proceedings of the ACM SIGMOD Conf. on Management of Data (SIGMOD-2001).
- [5] Google Inc., The Google API, <http://www.google.com/apis/>
- [6] Robert E. Schapire. A brief introduction to boosting. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.