

Hub&Authority Scoring Algorithm Project in Nutch

Yongqiang Li

1 Introduction of Scoring Algorithms in Nutch

The web page scoring algorithms are classified into off-line algorithm and on-line algorithm by the data visiting styles. And they are also can be classified into content based algorithm and link structure based algorithm by the scoring strategies.

1.1 Off-line Scoring Algorithms

1.1.1 TFIDF Algorithm

TF-IDF scoring algorithm is based on analyzing the term frequency (TF) over the inverse document frequency (IDF). This is a page content based algorithm.

1.1.2 WebGraph and Link Analysis

<http://wiki.apache.org/nutch/NewScoring> LinkRank is a PageRank-like algorithm[1] to score all the web pages in the web graph by using iterative processes and converge to the final scores for all web pages. LinkRank has some optimized handling to ignore those page links inside the same domain of a web page and omit reciprocal page links between web pages(not very clear, only count once?). Basically this algorithm is different from OPIC which is an on-line algorithm. We can see this off-line scoring algorithm is based on the link structure of web pages. This method is different from the content based scoring algorithm.

1.2 On-line Scoring Algorithms

1.2.1 OPIC algorithm in Nutch

<http://wiki.apache.org/nutch/FixingOpicScoring> Adaptive On-line Page Importance Calculation algorithm is an on-line scoring algorithm[2]. This scoring algorithm uses cash to measure the importance of a webpage. Initially every web page has an equal amount of cash. The importance of a webpage is summed by historical cash and the current cash. The historical cash is the cash which flows into the web page last time. Whenever a web page is processed, its cash flows into its outer links and the current cash is deposited into the historical cash. Then the current cash is reset to zero. This algorithm is only partially implemented in Nutch for some problems. It seems that this algorithm is not complete and leading those problems.

1.3 Problems and Solutions

Generally the on-line algorithm is more challenging than off-line algorithm for the real-time page updating or processing. And the content based algorithm is more efficient but not that reliable. Because it is easier to analyze the term occurrence than the link graph among all web pages. But the content based algorithm is more easily to be cheating by putting more key words into one page. Then the link structure based algorithm is more reliable. Still this page link based algorithm faces the cheating challenges of link farms.

In Nutch the major scoring method is the TFIDF algorithm. Also a recent plug-in of WebGraph and Link Analysis based on link structure is implemented. In addition the OPIC algorithm is implemented partially for many pending problems. Because none of content based algorithm and link structure based algorithm are complete. As the Professor advised in the lecture, it is possible to find a way to combine those two methods together.

And the scoring algorithm of hubs and authorities is such an algorithm[3]. This algorithm first utilizes the high efficient content based algorithm to filter the top 200 web pages as the root set. Then expand the root set to 5,000 base set. In the end the algorithm of analyzing hubs and authorities based on link structure is utilized to score the highest ranking pages. We can see this method is very efficient and effective. For people only care limited number of web pages.

Still the off-line scoring algorithm is behind the latest web pages several days. It is promising to utilize the on-line algorithm.

2. Related Work

2.1 Scoring Architecture Design

It's very fortunate that the Nutch 1.1 has implemented the scoring algorithm of WebGraph. The WebGraph does not use the Nutch plugin architecture. It is made to work within the framework of the FieldIndexer. The idea was that Nutch are supposed to have many different scoring algorithms to a final score for a document in the index. These different algorithms would be implemented as MapReduce jobs that created some output. That output would then be turned into Field objects, usually with a type of Document Score. These Field objects would then be aggregated by the FieldIndexer to create the final document score in the index. The LinkRank job is an example of just such an algorithm. It uses the WebGraph job's output of Inlinks, Outlinks, and Nodes.

In the case of WebGraph and LinkRank the link analysis score is actually put back into the CrawlDb using the ScoreUpdater job. It is then pulled into a BOOST type Field object in the BasicFields job. From there the FieldIndexer and the FieldFilter plugins create Lucene index fields from the Nutch Field objects. The boost field filter, field-boost, is what actually takes the boost fields including the linkrank and aggregates them together for a final score which is set as the document boost in the index.

The FieldIndexer architecture is simple though. It take multiple inputs, each of which is a Text(url) -> Field sequence or map file. These fields are aggregated by url and then passed to field filter plugins in the reduce phase. The plugins are responsible for taking the values out of the FieldWritable and FieldsWritable objects, translating them to Lucene field objects, and putting them into the Lucene Document. This is then written to the index in the OutputFormat of the FieldIndexer. See the field-basic and field-boost plugins for examples of how this works.

For the new scoring algorithm of hub&authority, a new MapReduce job is needed. The WebGraph output could also be used as input into this new job as all of inlinks, outlinks, and nodes for urls are needed. Have the new job create some output such as url -> score. Then it is to create a MapReduce job similar to BasicFields or AnchorFields that turns that into a Field object of type BOOST. This would then be input into the FieldIndexer to create the final output. It is also needed to take the approach of taking the new final score by url and putting that back into the CrawlDb, similar to how the ScoreUpdater does currently.

2.2.1 Hub and Authority Approach Design

For the new scoring algorithm of hub and authority, a new HubAuthority job of MapReduce job is designed and implemented. The WebGraph output is used as input into this new job as all of inlinks, outlinks, and nodes for urls are needed. The entire class design of HubAuthority is shown in Figure 1.

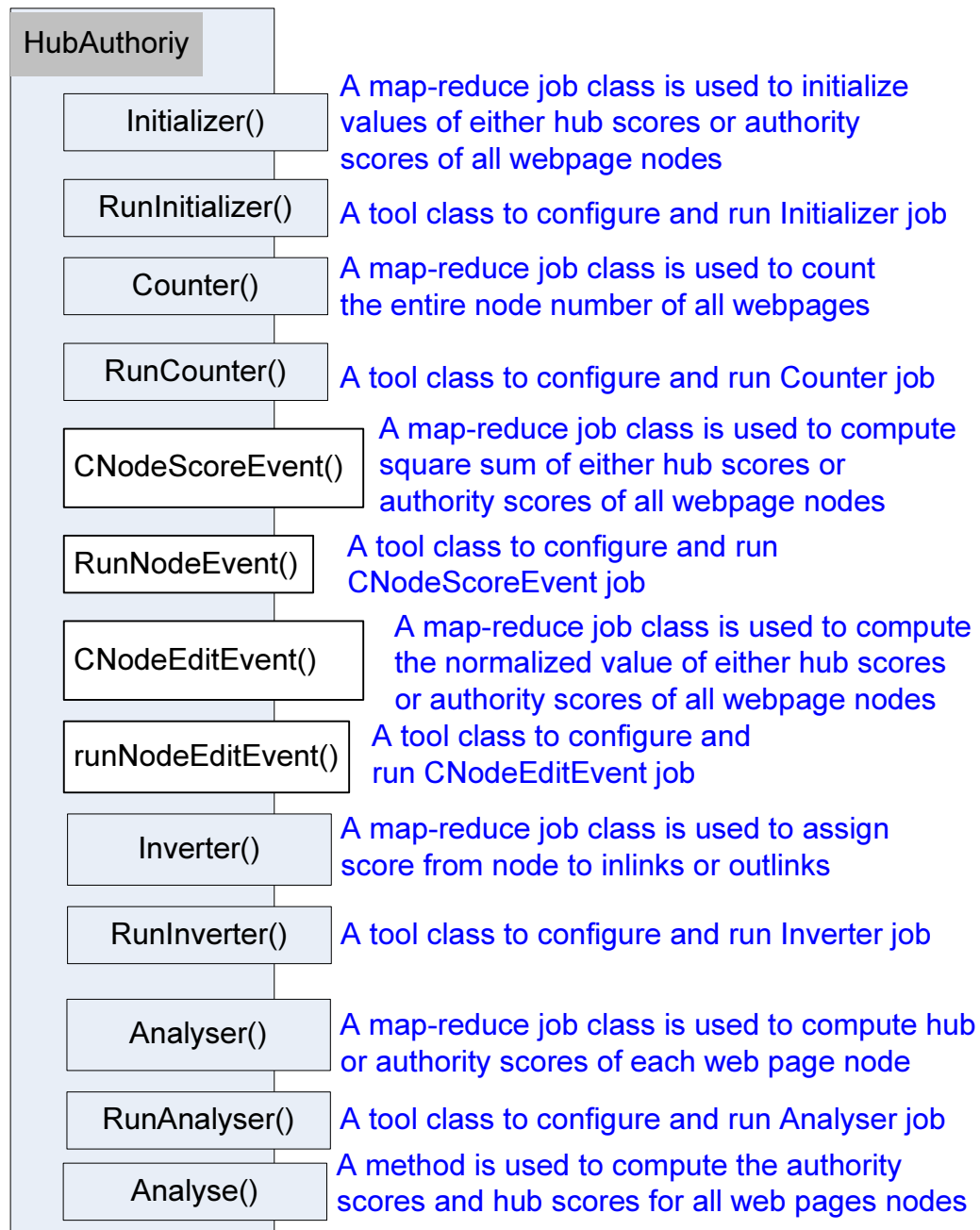


Figure 1. the HubAuthority class

2.2.2 Data Structure of a Node in the Hub and Authority Approach

Since there are two scores in a node in the hub and authority approach, the current design with one score can not satisfy the requirements. And the Node class is upgraded by adding a new member variable as shown in Figure 2.

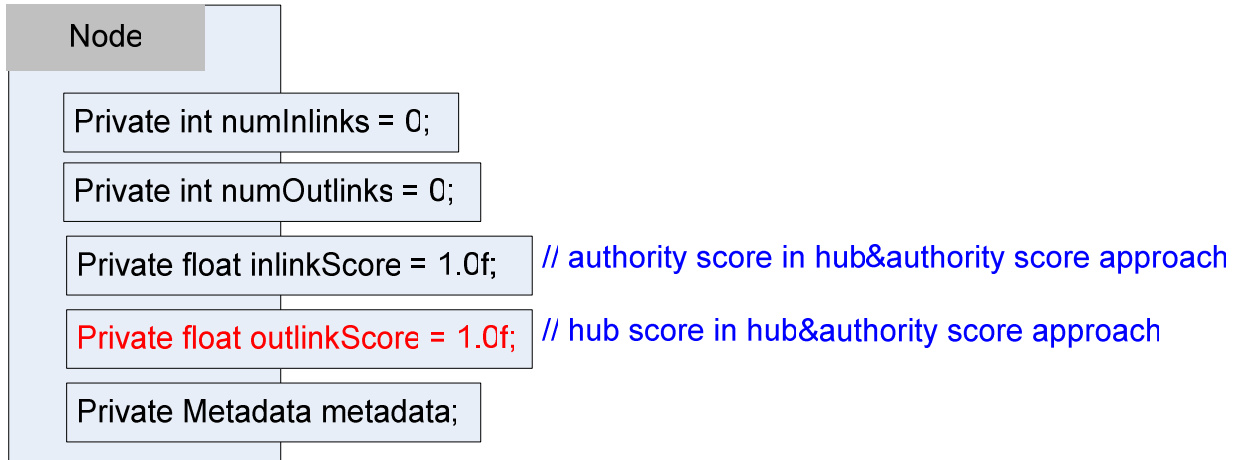


Figure 2. Upgraded class of Node by adding a new member outlinkScore

2.2.3 Iterative Hub and Authority Algorithm

The iterative computation process is the same as that in [1] shown in Figure 3.

```

Iterate( $G, k$ )
   $G$ : a collection of  $n$  linked pages
   $k$ : a natural number
  Let  $z$  denote the vector  $(1, 1, 1, \dots, 1) \in \mathbb{R}^n$ .
  Set  $x_0 := z$ .
  Set  $y_0 := z$ .
  For  $i = 1, 2, \dots, k$ 
    Apply the  $\mathcal{I}$  operation to  $(x_{i-1}, y_{i-1})$ ,
    obtaining new  $x$ -weights  $x'_i$ .
    Apply the  $\mathcal{O}$  operation to  $(x'_i, y_{i-1})$ ,
    obtaining new  $y$ -weights  $y'_i$ .
    Normalize  $x'_i$ , obtaining  $x_i$ .
    Normalize  $y'_i$ , obtaining  $y_i$ .
  End
  Return  $(x_k, y_k)$ .

```

Figure 3. Upgraded class of Node by adding a new member outlinkScore

2.2 Focused subgraph generation

Figure 4 shows how to generate a focused subgraph set of web page links.

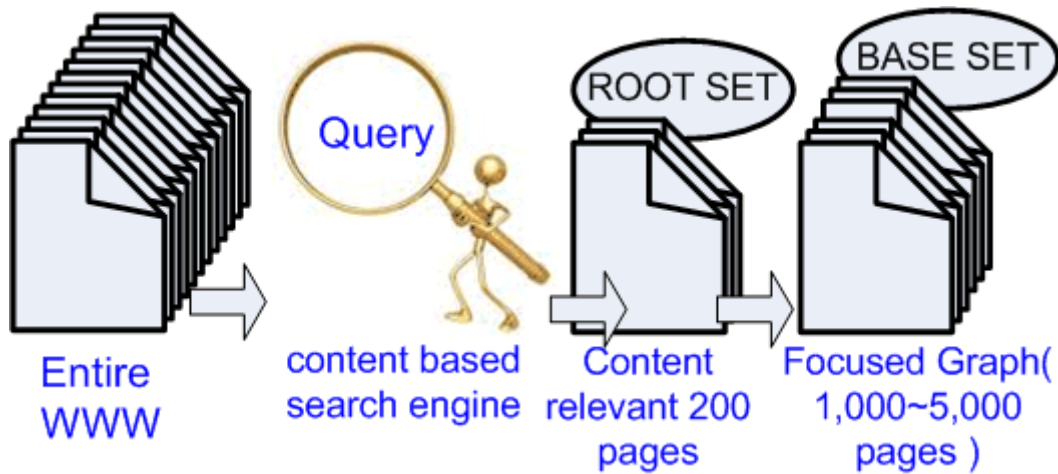


Figure 4. how to build a focused subgraph of entire web pages

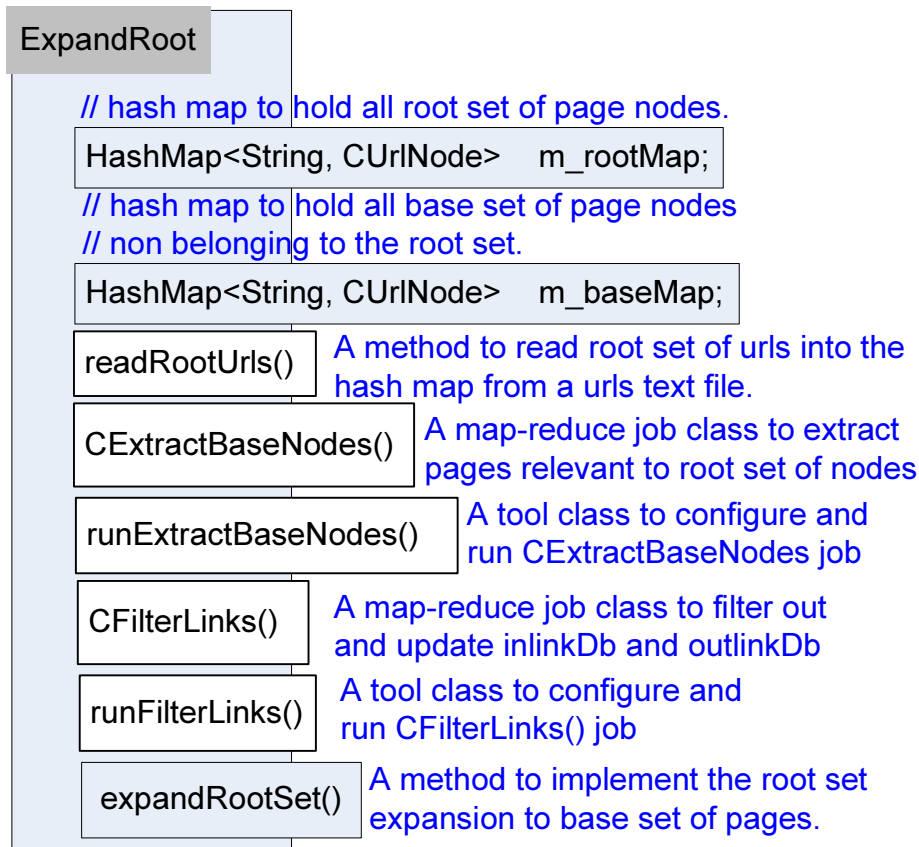


Figure 5. the ExpandRoot class

In the implementation all urls relevant to a specific query are found by using the search engine. These root set of urls are collected and saved into a text file. Then a class ExpandRoot (shown in Figure 5.) is used to read the root set of urls file into the memory and saved into a hash map data structure. Then a Map-Reduce job of CExtractBaseNodes is used to traverse all outlinkDb obtained from WebGraph and extract out all nodes relevant to the root set of urls. All these extracted relevant nodes and original root set of urls nodes are put into the base set. Next another Map-Reduce job of CFilterLinks is used to filter out all non-relevant links and

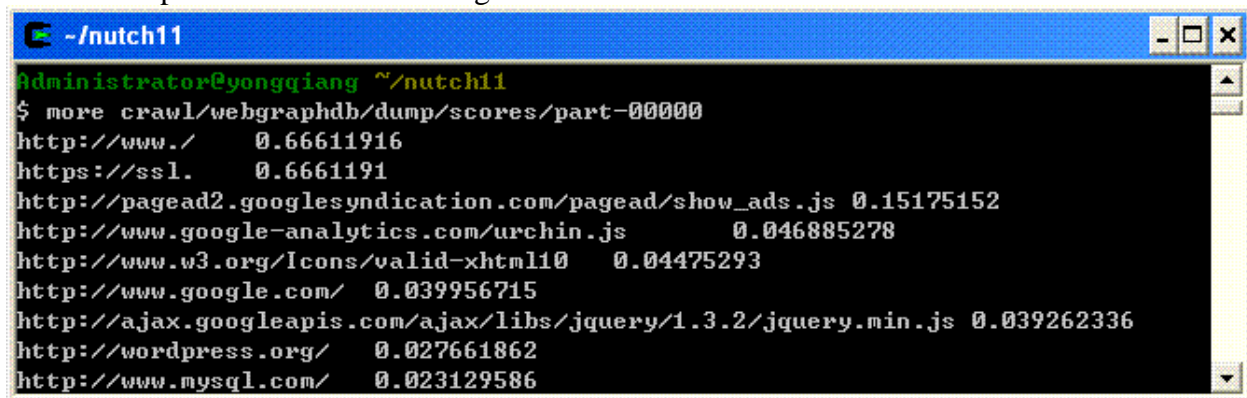
update the outlinkDb and inlinkDb. OutlinkDb is a set of links with keys from which the link is pointing to another url. InlinkDb is a set of links with keys to which the link is pointed from another url.

3. Test Results

The total number of crawling pages are 50,000. This section is to test the Hub&Authority scoring algorithm for some broad topic queries.

3.1 Query “google”

The top results are shown in Figure 6.

A terminal window titled '-/nutch11' showing the output of a 'more' command. The output lists several URLs with their corresponding scores. The top two results are 'http://www./' and 'https://ssl.', both with a score of 0.66611916. The third result is 'http://pagead2.googleadsyndication.com/pagead/show_ads.js' with a score of 0.15175152. The fourth is 'http://www.google-analytics.com/urchin.js' with a score of 0.046885278. The fifth is 'http://www.w3.org/Icons/valid-xhtml10' with a score of 0.04475293. The sixth is 'http://www.google.com/' with a score of 0.039956715. The seventh is 'http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js' with a score of 0.039262336. The eighth is 'http://wordpress.org/' with a score of 0.027661862. The ninth is 'http://www.mysql.com/' with a score of 0.023129586.

```
Administrator@yongqiang ~/nutch11
$ more crawl/webgraphdb/dump/scores/part-00000
http://www./      0.66611916
https://ssl.     0.6661191
http://pagead2.googleadsyndication.com/pagead/show_ads.js 0.15175152
http://www.google-analytics.com/urchin.js      0.046885278
http://www.w3.org/Icons/valid-xhtml10         0.04475293
http://www.google.com/                        0.039956715
http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js 0.039262336
http://wordpress.org/                         0.027661862
http://www.mysql.com/                         0.023129586
```

Figure 6. the query results of “google”

The top 2 results are two invalid urls, caused by invalid outlinks. When I hard coded to filter out these two invalid urls, the scoring results are not as reasonable as above results. And some other unexpected pages like *.js should be filtered out. Anyway we still find “www.google.com” is on the top 5 search results.

3.2 Query “manufacturer”

Figure 6 shows the query results of “manufacturer”. It is very reasonable to notice that many well-known manufacturers are listed on the top ones.

```
-/nutch11
Administrator@yongqiang ~/nutch11
$ more crawl/webgraphdb/dump/scores/part-00000
https://ssl.      0.4383241
http://www./      0.4383241
http://www.pacetech.com/      0.08946074
http://www.usa.canon.com/      0.08946074
http://www.vinten.com/      0.08946074
http://www.zoome.com/      0.08946074
http://www.tiffen.com/      0.08946074
http://www.telemetryinc.com/      0.08946074
http://www.aaton.com/      0.08946074
http://www.steadytracker.com/      0.08946074
http://www.sony.com/      0.08946074
http://www.anvilcase.com/      0.08946074
http://www.apgcases.com/      0.08946074
http://www.arri.com/      0.08946074
http://www.avvideo.com/      0.08946074
http://www.barbizon.com/      0.08946074
http://www.bogenphoto.com/      0.08946074
http://www.sekonic.com/      0.08946074
http://www.bulbtronics.com/      0.08946074
http://www.chapman-leonard.com/      0.08946074
http://www.chimeralighting.com/      0.08946074
http://www.chyron.com/      0.08946074
http://www.citytheatrical.com/      0.08946074
http://www.claypaky.it/      0.08946074
http://www.clearcom.com/      0.08946074
http://www.colortran.com/      0.08946074
http://www.cool-lux.com/      0.08946074
http://www.da-lite.com/      0.08946074
http://www.dar.uk.com/      0.08946074
http://www.dedolight.com/      0.08946074
http://www.panasonic.com/broadcast/      0.08946074
http://www.freefind.com/      0.08946074
http://www.fujifilm.com/      0.08946074
http://www.glidecam.com/      0.08946074
http://www.rosco.com/      0.08946074
http://www.hdal.com/      0.08946074
http://www.highend.com/      0.08946074
```

Figure 6. the query results of “manufacturer”

3.3 Score all urls

Figure 7 shows the scoring results of all urls.

```
Administrator@yongqiang ~/nutch11
$ more crawl/webgraphdb/dump/scores/part-00000
http://www./ 0.7049385
https://ssl. 0.70454425
http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab 0.047943
313
http://pagead2.googlesyndication.com/pagead/show_ads.js 0.042457804
http://www.adobe.com/go/getflashplayer 0.02826939
http://www.google-analytics.com/urchin.js 0.013369016
http://s7.addthis.com/js/250/addthis_widget.js 0.009097187
http://www.google.com/jsapi 0.0079745585
http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js 0.007901392
http://www.addthis.com/bookmark.php 0.0072559174
http://validator.w3.org/check/referer 0.0066658184
http://jigsaw.w3.org/css-validator/check/referer 0.006526673
http://www.facebook.com/home.php? 0.006352804
http://search.dntracker.com/tags/us.js 0.0060321754
http://www.macromedia.com/go/getflashplayer 0.0058546043
http://edge.quantserve.com/quant.js 0.005515658
http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js 0.005379929
http://www.joomla.org/ 0.005097794
http://widgets.twimg.com/j/2/widget.js 0.005094266
http://lite.piclens.com/current/piclens_optimized.js 0.005019684
http://s7.addthis.com/js/200/addthis_widget.js 0.0046877502
http://www.facebook.com/ 0.0043270555
http://wordpress.org/ 0.0040557953
http://www.liveinternet.ru/click 0.0037483566
http://partner.googleadservices.com/gampad/google_service.js 0.0037154397
--More--(0%)
```

Figure 6. the query results of “manufacturer”

We noticed some popular website, such as www.google.com and www.facebook.org are listed above.

4. Conclusions

From this course project of hub&authority scoring algorithm implementation in Nutch 1.1, it is noticed that this hub&authority is a promising scoring algorithm to score those broad topic query. Since this scoring approach is based on link structure approach, it requires the base set of nodes to have dense enough links. Many pages are out linking to some advertisements and non-relevant pages to the query. In my course project only 50,000 urls are crawled, it needs to crawl the entire web to verify the effectiveness of this hub&authority algorithm. Currently this hub&authority scoring algorithm is a little slow.

5. Acknowledgement

Here I want to acknowledge Professor Mattmann, I could not achieve the current status without his guide. Also I want to acknowledge committer Dennis Kubes, without his instructional explanation I can not obtain above design of the scoring architecture for the authorities and hubs approach.

References:

- [1] S. Brin, L. Page, "Anatomy of a Large-Scale Hypertextual Web Search Engine," *Proc. 7th International World Wide Web Conference*, 1998.
- [2] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proceedings of WWW Conference*, May 2003.
- [3] Kleinberg, J. M., Authoritative sources in a hyperlinked environment, *J. ACM* 46, 604–632 (1999).

Appendix:

Here is the checklist of the added new source code files and Nutch development version:

- Nutch version is version 1.1
- The newly added Class checklist is shown in Table 1.

Table 1		
Class Name	location	function
ExpandRoot	org.apache.nutch.scoring.webgraph.ExpandRoot	Generate a base set from a root set of urls and update the inlinkDb and outlinkDb
HubAuthority	org.apache.nutch.scoring.webgraph.HubAuthority	Iterate to use Hub&Authority approach to score the pages for a given broad topic query
QueryList	org.apache.nutch.searcher.QueryList	Generate a root set of urls for a given broad topic query

- The modified Class checklist is shown in Table 2.

Table 2		
Class Name	location	function
Node	org.apache.nutch.scoring.webgraph.Node	Data structure of a web page

- How to use commands of and QueryList , ExpandRoot and HubAuthority.

The usage is very similar to the linkrank of webGrpah as shown in

<http://wiki.apache.org/nutch/NewScoringIndexingExample?highlight=%28FieldIndexer%29>

Figure 4 shows the commandlines to use HubAuthority

```
$bin/nutch inject crawl/crawldb crawl/urls/
$bin/nutch generate crawl/crawldb/ crawl/segments
$bin/nutch fetch crawl/segments/20090306093949/
$bin/nutch updatedb crawl/crawldb/ crawl/segments/20090306093949/
$bin/nutch invertlinks crawl/linkdb -dir crawl/segments
$bin/nutch index crawl/indexes crawl/crawldb crawl/linkdb crawl/segments/*
$bin/nutch org.apache.nutch.searcher.QueryList query crawl/urls.txt
$bin/nutch org.apache.nutch.scoring.webgraph.WebGraph -segment
crawl/segments/20090306093949/ -webgraphdb crawl/webgraphdb
$bin/nutch org.apache.nutch.scoring.webgraph.ExpandRoot -webgraphdb
```

```
crawl/webgraphdb/ -rootset crawl/rootUrls.txt
$bin/nutch org.apache.nutch.scoring.webgraph.Loops -webgraphdb
crawl/webgraphdb/
$bin/nutch org.apache.nutch.scoring.webgraph.HubAuthority -webgraphdb
crawl/webgraphdb/
$bin/nutch org.apache.nutch.scoring.webgraph.ScoreUpdater -crawlddb
crawl/crawlddb -webgraphdb crawl/webgraphdb/
$bin/nutch org.apache.nutch.scoring.webgraph.NodeDumper -scores -topn 1000 -
webgraphdb crawl/webgraphdb/ -output crawl/webgraphdb/dump/scores
```

Figure 4. commandlines