



Sampling frequency tuning tool

Supervisor: Morten Goodwin Olsen

Group members: Qi Jin

Wei Sun

Yang Wu

Catalogue

Abstract.....	3
Executive Summary.....	4
Introduction.....	5
Background (Review and Literature).....	7
Problem Statements.....	10
Requirements specification for Crawler.....	12
Design of the crawler.....	13
Implementation.....	14
Evaluation and Testing.....	17
Discussion.....	19
1. Project Outcomes.....	19
2. Evaluation of the overall results.....	20
Conclusion.....	22
References:.....	24

Abstract

“The goal of this project is to find ways to optimize the crawler frequency for individual web sites. The idea is to avoid crawling a site in case the accessibility to the site has not been changed. This is a challenge for all search engines to focus the resources on actual changes. Another relevant aspect of sampling is to select a significant and representative set of sites.”

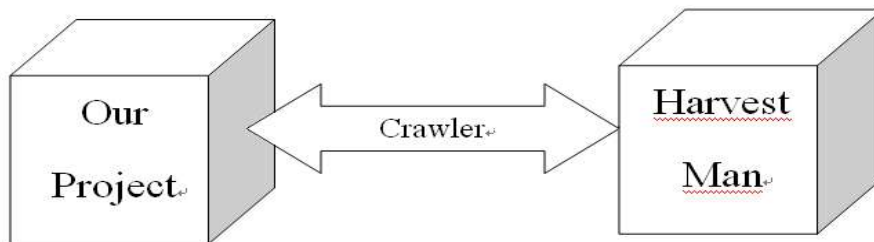
In this project, we are supposed to develop a real live crawler to monitor the accessibility change of each web page and return the update frequency to achieve the goal. And we are going to use the competitive game of learning automata theory because of its high effectiveness and fast speed, compare with Round Robin.

More information in our project web page: <http://www.jinqi.org>

Executive Summary

In our project, we are supposed to design a program which can monitor the update frequency of each website. Then let search engine visit the website with the highest frequency in order to save resource. We plan to use the competitive game of learning automata algorithm which bases on knapsack algorithm to solve this problem. And at last we can get the update frequencies of every websites which we monitored.

Here is a sketch map of our project:



Our project is an improvement and complement to HarvestMan. HarvestMan is an internet crawler (robot) program written in Python. It helps you to grab pages from the internet and store it in a local directory for offline browsing. Our project is a process about the recrawling, not the initial crawl. It means that the local collection is not empty but already has been fully filled by HarvestMan. And we use our algorithm to replace the Round Robin algorithm which be used in HarvestMan. Crawler is an interface which connects our job to HarvestMan. Accord to Crawler, our program monitors the websites accessibility changes and tries to detect the frequency of each website, and then tell HarvestMan to visit the website with the highest frequency firstly.

Introduction

Since the World Wide Web is the largest and most widely known repository of hypertext, it's very important to find out the very resource we wanted among the immense amount of information. In no doubt, search engines play an important role on it. But what are the components of the search engineer? Now days, almost all search engineer are using some large-scale programs that fetch tens of thousands of web pages per second. One of them, the crawler, is always utilized for monitoring whether the local copy of the data sources is the latest version of the authority.

How to keep it up-to-date? The traditional way is using a periodic crawler (also called round robin). It works as its name: just check the web site on the queue one by one. And then, repeat it. There is a more efficient way: using incremental crawler. The strategy of this kind of crawler is refreshing existing pages and replacing "less important" pages with new and "more important" pages in the queue.

For instance, if the crawler can estimate how often pages changed, it can revisit the only pages that have changed (with high probability, instead of refreshing the entire local collection altogether). Clearly, the effectiveness of crawling techniques heavily depends on how web pages change over time. If most pages change at similar frequencies, the periodic crawler and the incremental crawler may be equal effective.

Note that we assume that each web page has a given, possibly bias, change frequency. This is according to the findings in the literature [1]. Our job in this project is to find out each website's update frequency in order to get each one's priority. The web page changes we consider is the web page's accessibility change. And we measure the web page's accessibility change by monitor the HTML tags' change. Since it's a big job, and we've got only 3 members especially in one semester, we got a very good advice: using a crawler called harvestman. Although Harvestman is a completed crawler with all function enabled, but in that particular part, it's using some algorithm like Round Robin. So we can do some implements to his job. Since our project belongs to one part of the EIAO, we can just focus on the accessibility changes but not the entire

website. To simplify our job, we assume that each web page has a same time-cost on downloading (visiting), and we just consider the main page of each website, ignore the sub pages and hyperlinks, as done in earlier work in [1,5].

All of these works are pre-claimed. And if we still have enough time, we will do some further study on it. We will be glad to see our works can optimize that crawler indeed.

Here below is our research agenda:

1. make a task definition
2. search enough information about the project
3. have a comprehension about knapsack problem
4. monitor the changes of HTML tag
 - 4.1. have a comprehension about HTML
 - 4.2. search and filter some useful python applications as examples
 - 4.3. have a comprehension about .CSS
5. code and implement the algorithm with python
6. implement our application in HarvestMan

Background (Review and Literature)

In this section, we will present some relevant works briefly.

1. Effective Change Detection Using Sampling [1]

Junghoo Cho Alexandros Ntoulas

In this paper they propose three sampling-based download policies that can identify more changed data items effectively. In their sampling-based approach, they first sample a small number of data items from each data source and download more data items from the sources with more changed samples.

They also give out the advantages and disadvantages of the frequency-based policy:

_ **Advantage:** The frequency-based policy is proven to be optimal when we can estimate the change frequencies of data items accurately.

_ **Disadvantage:**

1) It is very difficult to estimate the change frequency of a data item accurately. Unless we have a long change history of a data item, existing estimation methods often lead to unreliable predictions, which in turn lead to an undesirable download policy. In addition, the change frequency itself may change over time, but we may not realize that it has changed.

2) In order to estimate the change frequencies, we need to keep track of the change history of every data item. When we maintain a large number of items, this tracking may incur significant storage and maintenance overhead

2. A Data-Mining Approach for Optimizing Performance of an Incremental Crawler [2]

M. K. Mohania IBM India Research Lab I.I.T., Hauz Khas

In this paper, author is interesting in telling us following two topics: The Update Score module frequency and how to use data-mining way to make it more efficient. The update score module frequency is to update the weights in the All URLs table. It is important to know with which frequency the module should update the database. And there are 3 ways:

Real Time update: The first method that comes to mind is to update the database as soon as they have a user who sends a query. This is the easy way but also the heavy solution. If the All URLs table is refreshed each time the user clicks on a link, they risk overloading the database. Thus this method is not really efficient.

Scheduled Time update: Another solution is to update the database after a certain time. The best solution can be to run the Update Crawl module at the end of the day when few people are using the search engine. The disadvantage of this method would be to refresh lately the pages which are the most asked by the user during the day.

Scheduled Limited update: The last solution is to specify a limit. When the quantity of information stored on the Users Data reaches the limit, the Update Score module will change the value of the weight on the database. Typically, it depends on the size of the collection and the frequency of visits of the users on the search engine website. Also, it is important to know, which data provided by the users, can help the crawler to improve the freshness of the collection. The goal is to revisit more frequently the pages which are the most visited by the users. So, using Usage-Mining in Data-Mining is an efficient way. But since it's not very related to our work, we won't give some further details about it.

3. iSurfer: A Focused Web Crawler Based on Incremental Learning from Positive

Samples [3]

Yunming Ye¹, Fanyuan Ma¹, Yiming Lu¹, Matthew Chiu², and Joshua Huang²

This paper presents a focused Web crawling system iSurfer for information retrieval from the Web. Different from other focused crawlers, iSurfer uses an incremental method to learn a page classification model and a link prediction model. It employs an online sample detector to incrementally distill new samples from crawled Web pages for online updating of the model learned. Other focused crawling systems use classifiers that are built from initial positive and negative samples and can not learn incrementally. The performances of these classifiers depend on the topical coverage of the initial positive and negative samples. However, the initial samples, particularly the

negative ones, with a good coverage of target topics are difficult to find. Therefore, the iSurfer's incremental learning strategy has an advantage. It starts from a few positive samples and gains more integrated knowledge about the target topics over time. Our experiments on various topics have demonstrated that the incremental learning method can improve the harvest rate with a few initial samples.

4. An Adaptive Model for Optimizing Performance of an Incremental Web

Crawler [4]

Jenny Edwards Kevin McCurley John Tomlin

This paper outlines the design of a web crawler implemented for IBM Almaden's WebFountain project and describes an optimization model for controlling the crawl strategy. This crawler is scalable and incremental. The model makes no assumptions about the statistical behavior of web page changes, but rather uses an adaptive approach to maintain data on actual change rates which are in turn used as inputs for the optimization. Computational results with simulated but realistic data show that there is no 'magic bullet' - different, but equally plausible, objectives lead to conflicting 'optimal' strategies. However, we find that there are compromise objectives which lead to good strategies that are robust against a number of criteria.

5. Incremental Web Crawling as a Competitive Game of Learning Automata [5]

Svein Arild Myrer Morten Goodwin Olsen

In this thesis authors address the new problem area of monitoring highly dynamic data sources of different importance. They use the concept of an incremental web crawler as a basis for the novel approach where we consider the incremental crawling task as a continuous learning problem where scheduling of monitoring tasks is combined with parameter estimation in an on-line manner. By mapping the problem to two variants of the so called knapsack problem they propose two solutions based on a machine learning technique known as learning automata.

Since the frequency of the websites' changing rate is a dynamic data source in the internet, this paper is the most important literary reference to our task. It not only provides us the solution of the project – using LA, but also gives us the specialization detail in our job. Maybe we can say that our job is an implement to their learning. We use the simplified version of their LA and try it in the real-world environment.

Problem Statements

In this section, we will present what we probably will face in designing our crawler.

First of all, we should know what the “accessibility” exactly means in this project. And how could we monitor it to see whether it has been changed. In [7], author states some aspects seem relevant to a number of different perspectives. To our project, we should concern “Access to content for automatic processing, such as search engine indexing or automatic reporting from local authorities to central government”. That is

to say, to report the accessibility changes to search engine. But how could we do this? As we know, the web-mining can be divided into three parts: Web content mining, Web structure mining and Web usage mining. Since the crawler concerns more about the Web content mining, shall we focus on the same region? Or maybe we should pay more attention in Web usage mining in order to get the responds from users to check out whether their behavior imply some accessibility changes. Note that we assume that the accessibility changes are in the HTML-tags only. This is according to the Web Content Accessibility Guidelines [7] which claims that most accessibility issues are connected to tags. We should know it before us starting our task.

The second one is about the algorithm of the crawler. The Round-Robin policy is currently being used by many systems due to its simplicity. But the disadvantage is obviously: wasting of the search engine’s bandwidth. Round robin algorithm major strategy is to visit each site the same amount. For example: there is 3 sites: {s1,s2,s3}, and if the queue is {s3,s2,s1}, the Scheduling will be:

Site:	s3	s2	s1	s3	s2	s1	s3	s2	s1
Time:	t1	t2	t3	t4	t5	t6	t7	t8	t9

It doesn’t care if the webpage has been updated. So if the website s3 only changes per 9 time sort, there will be $9/3-1=2$ time sort searching bandwidth/resource are wasted. How could we do to avoid that waste? There is another algorithm called focused crawler. With focused crawling, if it is determined that a page is not relevant or its links should not be followed, then the entire set of possible pages underneath it are pruned and not visited. Can it be used in monitoring accessibility changes? That’s the major problem we want to resolve.

The third one will be the difference between our job and previous work. In [5], author only tested LA in a virtual environment, not in a real internet world. So in our task, we will try this LA algorithm in a small circumstance. Beside that, we will also use a simple version of LA where we do not consider the update rate of page weighted against its importance value. We do assume that all the pages get same important value and the cost of downloading each page will be same. That is to say, in our case, one word web page will probably equals www.cnn.com ‘s index page though there will be huge difference.

Requirements specification for Crawler

Compare with other crawler's strategy, say Round-Robin, the advantage of our crawler is obviously. But how about compare with same strategy's crawler? What kind of special require should we use? Let's take a look at what kind of LA used in <Incremental Web Crawling as a Competitive Game of Learning Automata> and ours:

C.G.L.A. in previous work:	Our LA:
2 kind of LA: binary and fractional.	Fractional only.
Use knapsack problem as an algorithm.	Use knapsack problem as an algorithm.
Use Update rate of page weighted against its importance value as each website's value.	Use update rate of page only.
Use Cost of polling web page as volume of item.	Consider each web page has a same volume of item.

There is some reason for us to use the simple version of the existed crawler: firstly, it's because our knowledge is not so professional. What's more is that we don't have enough time for this design. Last factor is that our job is only a little test in real world based on previous' work.

Beside the algorithm, it's also important to specify requirements of our designed crawler. It should include following function:

1. Use harvestman to download files (only each page's "index.html").
2. Read each website's index file and assign it to LA.
3. Algorithm.
4. Output.

Design of the crawler

First, we introduce the Round Robin algorithm briefly, which visit each site the crawler will detect with the same amount. E. G .There are 3 sites: (s1, s2, s3), the previous crawler visits the sites in a circular order like this: first s1---s2---s3, then s1---s2---s3

It wastes an amount of system recourses, so we want to make some changes to the original crawler. In our program, we use the knapsack algorithm, which visits based on update probability.

We assume each web page has the equal download time ,and give a value of $F_j(n)$ divided by N to the variable S_{ij} , which represents the page j 's update probability in time slot i .

Here is the procedure:

each ti {t1 ,t2, t3, T4}

each page j

r =random (0,1)

s_{ij} =F_j (n)/N

if r <s_{ij} :

download page j

else :

do nothing

if page j updated and not exceeded :

F_j (n)++

else not updated or exceeded :

F_j (n)- -

For each page. At each time slot, we give them the same random value generated by random function and ranged between 0 and 1.

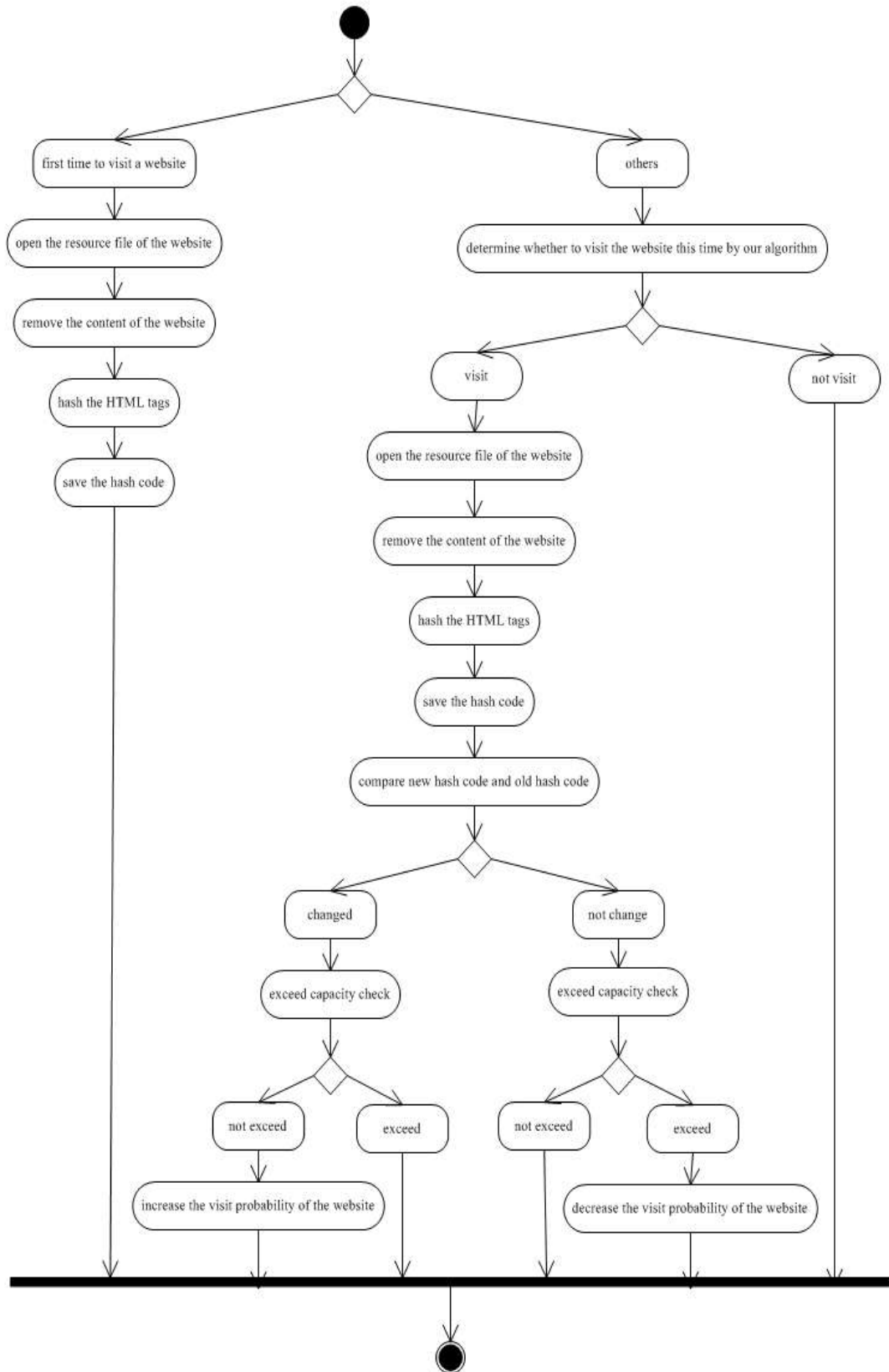
Then we will check if r is greater than or less than S_{ij} , if r is less than S_{ij} , we download the page, else do nothing.

Then we detect if the page has changed, if S_{ij} updated then $F_i(n)$ add one (e. g. $F_i(n) / N$ changes from $3/5$ to $4/5$) else $F_i(n)$ changes from $3/5$ to $2/5$)

Of course $F_i(n)/N$ should range between 0 and 1, so if it exceed capacity $F_i(n)$ subtract one, if it does not exceed $F_i(n)$ add one. Due to smaller parameter N faster, less accurate, but larger N slower, more accurate. So we will choose the proper value through our later experiment.

Implementation

Here is the flow chart of our main program.



When the first time to visit a website, our program asks crawler to download the website into locate collection, and then the program open the resource file of website in the locate collection. As we only monitor the accessibility changes of website, not the content changes, so the program removes the content of the website. Then the

program hashes the HTML tags of website and save the hash code.

When the second time, third time and so on to check the same website in the Round Robin sequence, the main algorithm in our project will determines whether to visit it now. If the website has a high visit probability, we visit it now, otherwise we skip it. How the main algorithm works will be shown in detail by my group member later. When the program decided to visit the website, it open the resource file of the website, remove the content, hash the HTML tags and save the hash code, this part of work is the same with what it did to visit the website at the first time. Moreover, the program compares the new hash code with the old hash code. If the hash code changed, it means the accessibility of the website changed. And the program continue to check whether the amount of websites which wait for update exceed the crawler's capacity. Thus, there are four situations:

1. hash code changed and capacity was exceeded: do nothing
2. hash code changed but capacity was not exceeded: increase the visit probability of the website
3. hash code not changed and capacity was not exceeded: do nothing
4. hash code not changed but capacity was exceeded: decrease the visit probability of the website

In this way, we can give a feedback to crawler that the website with a higher visit probability should be visited earlier.

Evaluation and Testing

We tested our program with different timeslot. And results are hereby.

Timeslot = 100

First time test:

Number of internal visits **47** number of update number **6** in website www.cnn.com with probability **0.216666666667**

Number of internal visits **54** number of update number **12** in website www.hia.no with probability **0.3**

Number of internal visits **60** number of update number **18** in website www.bbc.com with probability **0.4**

Second time test:

Number of internal visits **99** number of update number **73** in website www.cnn.com with probability **1.0**

Number of internal visits **54** number of update number **13** in website www.hia.no with probability **0.45**

Number of internal visits **13** number of update number **1** in website www.bbc.com with probability **0.233333333333**

Third time test:

Number of internal visits **28** number of update number **6** in website www.cnn.com with probability **0.116666666667**

Number of internal visits **22** number of update number **2** in website www.hia.no with probability **0.0833333333333**

Number of internal visits **25** number of update number **6** in website www.bbc.com with probability **0.166666666667**

Timeslot = 50

First time test:

Number of internal visits **13** number of update number **10** in website www.cnn.com with probability **0.45**

Number of internal visits **13** number of update number **5** in website www.hia.no with probability **0.283333333333**

Number of internal visits **10** number of update number **1** in website www.bbc.com with probability **0.2**

Second time test:

Number of internal visits **16** number of update number **11** in website www.cnn.com with probability **0.45**

Number of internal visits **11** number of update number **1** in website www.hia.no with probability **0.2**

Number of internal visits **12** number of update number **3** in website www.bbc.com with probability **0.25**

Third time test:

Number of internal visits **22** number of update number **7** in website www.cnn.com with probability **0.416666666667**

Number of internal visits **21** number of update number **4** in website www.hia.no with probability **0.333333333333**

Number of internal visits **21** number of update number **4** in website www.bbc.com with probability **0.333333333333**

Timeslot = 10

First time test:

Number of internal visits **10** number of update number **7** in website www.cnn.com

with probability *0.966666666667*

Number of internal visits *9* number of update number *1* in website www.hia.no with probability *0.883333333333*

Number of internal visits *9* number of update number *1* in website www.bbc.com with probability *0.866666666667*

Second time test:

Number of internal visits *7* number of update number *4* in website www.cnn.com with probability *0.75*

Number of internal visits *6* number of update number *1* in website www.hia.no with probability *0.666666666667*

Number of internal visits *6* number of update number *1* in website www.bbc.com with probability *0.666666666667*

Third time test:

Number of internal visits *5* number of update number *3* in website www.cnn.com with probability *0.283333333333*

Number of internal visits *4* number of update number *1* in website www.hia.no with probability *0.233333333333*

Number of internal visits *4* number of update number *1* in website www.bbc.com with probability *0.233333333333*

Discussion

1. Project Outcomes

About what we have done is the program can detect the webpages if they are updated or not that we want the crawler to check out .and the program will give us a feedback and also show the update probability.

But we don't think about all the webpage's links just focus on "the index.html" of every page ,That makes our job more easy to implement .We saved the index file in to

the “c: \\downloads \\url \\url \\” folder by harvestman ,and then removed all the contents of the webpage ,left all the tags. Then we compare if the structure of the webpage has changed and get the result we mentioned above.

Which points we failed and why,, although we implement the RoundRobin algorithm and our Knapsack algorithm, are due to the program run doesn't work well ,and also a little slow ,so we can't test the program compared with the previous one to amount of webpages in the real world .Just compare both algorithms in a limited situation.

So for the future paths of development, what we should is to make our program run more efficiently so that it can be used into practice.

Here is the part if the output extracted from the results e. g.:

```
starting to download www.tianyaclub.com
updated
1.0
starting to download www.cnn.com
updated
0.783333333333
starting to download www.hia.no
done nothing
0.416666666667
starting to download www.bbc.com
not updated
0.583333333333
number of internal visists 8 www.tianyaclub.com
number of internal visists 7 www.cnn.com
number of internal visists 1 www.hia.no
number of internal visists 1 www.bbc.com
```

From above you can see, we tell the crawler to detect the webpages ten times, “done nothing” means not download. ”not updated” means the webpage hasn't changed although the crawler download it. Because from the prior results the crawler learned these two webpages don't change frequently, so they have a low visit probability that will led them less visited, on the other hand another two webpages will have high visit frequency.

2. Evaluation of the overall results

We think the reason about the program is too slow and sometimes don't work well is

may be some webpage is complicated so it will spend too much time in the download and remove content of webpage processes, which often make the process hangs. when we can just put the program into c:\windows\system32.cmd.exe it work better, seldom hangs happen although we still don't know clearly why it hangs when it runs in IDLE. We also try to use some other extension tool to speed the program like psyco, but it seems help few. According the situation we said above, we can reckon our assumption of equal cost of download is not true, but some web page like www.cnn.com need more system resource occupied and our program running slow mostly depends on Harvestman download speed. At the beginning in our algorithm, when the clawer detects web pages have changed we do self.f+1 (not +2),and when web page not updated we do self.f-1(not -2),through these way we can obviously show the user the probabilities if web pages have changed or not. It sounds good, but in fact, we found we use "self.f+2" this means seems better. And we also create a log.txt to note process for user to check the results more clearly.

Here is the result come from crawler detect www.tianya.com www.cnn.com
www.hia.com www.bbc.com in 10 times.
number of internal visists : 0 www.tianyaclub.com
number of internal visists : 5 www.cnn.com
number of internal visists : 1 www.hia.no
number of internal visists : 1 www.bbc.com

The last detected data returned by program:
starting to download www.tianyaclub.com
done nothing
0.116666666667

starting to download www.cnn.com
not updated
0.966666666667

starting to download www.hia.no
done nothing
0.383333333333

starting to download www.bbc.com
done nothing
0.333333333333

So we can see not matter the last weather www.cnn.com has changed, it has a high probability cause it has 5 times update in 10 times tests. So do the others.

Conclusion

In our algorithm , after some time slots, the program will automatically take more attention on the highly updated webpage ,download these pages ,and crawler will use its resource efficiently .That basically achieve our project purpose. And compared

with the Round-Robin algorithm, we can see the crawler has more visits to high updated webpage though our competitive game of learning automata instead of give average energy for each page.

There are also some bad results which don't match our assumptions. These will because of some design flaw of our software. Hope to get a chance to fix it.

References:

- [1]: Junghoo Cho, Alexandros Ntoulas,
Effective Change Detection Using Sampling
- [2]: M. K. Mohania, Hauz Khas,
A Data-Mining Approach for Optimizing Performance of an Incremental Crawler
- [3]: Yunming Ye, Fanyuan Ma, Yiming Lu, Matthew Chiu, and Joshua Huang,
iSurfer: A Focused Web Crawler Based on Incremental Learning from Positive Samples
- [4]: Jenny Edwards, Kevin McCurley, John Tomlin,
An Adaptive Model for Optimizing Performance of an Incremental Web Crawler
- [5]: Svein Arild Myrer, Morten Goodwin Olsen,
Incremental Web Crawling as a Competitive Game of Learning Automata
- [6]: Mikael Snaprud,
Accessibility to the internet
http://www.etsi.org/cce/proceedings/7_3.htm
- [7]: Caldwell B, Chisholm W, Vanderheiden G, White J (eds),
Web Content Accessibility Guidelines 2.0,
<http://www.w3.org/TR/2004/WD-WCAG20-20041119/>