The Design and Implementation for a C# Web spider

Ebada A. Sarhan *Iraky H. Khalifa*
Faculty of Computers & Information
Helwan University

Wael S. Tawah
Faculty of Computers & Information
Zagazig University

Abstract

It would be entirely too large a job for human workers to index and categorize the entire World Wide Web. This is a job that is almost always reserved for Web spiders. A Web spider is an automated program that searches the Internet for new Web documents and indexes their addresses and content-related information in a database, which can be examined for matches by a search engine. In this paper, we discuss the subject, present the design, and show how a Web spider can be written in C#.

1. Introduction

As the World Wide Web (Web) based Internet services become more popular, information overload also becomes a pressing research problem. Difficulties with searching on Internet get worse as the amount of information that available on the Internet increases. The number of indexable pages on the Web has exceeded 2 billion and is still growing at a substantial rate [1]. There are many different ways to search the Web automatically using programs which have been known by a variety of names: Spiders, Crawlers, Cruiser, Octopus, and Walker, Web robots, Web agent, Webbots, Wanderers, and Worms.

This paper demonstrates how to build a Web spider. A Web spider is an automated program that searches the Internet for new Web documents and indexes their addresses and content-related information in a database, which can be examined for matches by a search engine [2]. Web Spiders are generally considered to be a type of bot, or Internet robot. The primary difference between a spider and a simple bot is that the spider is capable of moving to new pages not originally requested by its programmer. Spiders proved to be very useful to one of the first utility sites to appear on the Web—the search engine.

It would be entirely too large a job for human workers to index and categorize the entire Web. This is a job that is almost always reserved for Web spiders, which scan websites and index their content. As the spider scans the site, it also looks at other pages to which the current site is linked. The spider keeps a list of these, and when it is finished scanning the current site, it visits these linked sites. Due to the widespread use of hyper-linking on the Web, it can be assumed that by following this pattern, a spider would eventually visit nearly every public page available on the Web.

In addition to performing indexing functions for search engines, spiders can scan a website looking for broken links, they can download the entire contents of a website to local hard drive, and they can also create a visual map that shows the layout of a website. Spiders prove useful whenever data must be retrieved from a site whose structure is not known beforehand by the programmer.

There are spiders written in Java, C++, Perl, and ASP [3]. We are not aware of any spiders written in C#, and that was one of the motivations behind writing this paper.

2. Literature Review

At its beginning as the ARPANET, the Internet was conceived primarily as a means of remote login and experimentation with telecommunication [4]. However, the main usage quickly becomes e-mail communication. This trend continues into the present form of the Internet, but with increasingly diverse support for collaborative data sharing and distributed, multimedia information access, especially using the Web. Many people consider the Internet and the Web the backbone of information superhighway and the window to the cyberspace.

The Web was developed initially to support physicists and engineers at CERN, the European Particle Physics Laboratory in Geneva, Switzerland [5]. In 1993, when several browser programs became available for distributed, multimedia, hypertext-like information fetching, Internet became the preview for a rich and colorful information cyberspace. However, as Internet services based on Web have become more popular, information overload has become a pressing research problem. The user interaction paradigm on Internet has been shifted from simple hypertext-like browsing to content-based searching. Many researchers and practitioners have considered Internet/Intranet searching to be one of the more pressing and rewarding areas of research applications.

Research in spiders began in the early 90's, shortly after the Web begins to attract increasing traffic and attention. Wanderer, written n 1993, was claimed to be the first spider for the Web [6]. Many different versions of spiders have since been developed and studied. Three major spider research directions have been developed and experimented with: speed and efficiency, spidering policy, and information retrieval [7].

3. Applications of Web Spiders

Spiders have been shown to be useful in various Web applications. There are four main areas where spiders have been widely used:

1- *Personal search*. Personal spiders try to search for Web pages of interest to a particular user. Because these spiders usually run on the client-machine, more computational power is available for the search process and more functionalities are possible [8, 9].

2- *Building collections*. Web spiders have been extensively used to collect Web pages that are needed to create the underlying index of any search engine [10]. In addition to building search engines, spiders can also be used to collect Web pages that are later processed to serve other purposes. For example, [11] used a spider to crawl the Yahoo hierarchy to create a lexicon of 400,000 words; [12] used Mercator to do random URL sampling from the Web; many others have used spiders to collect email addresses or resumes from the Web.

3- *Archiving*. A few projects have tried to archive particular Web sites or even the whole Web [13]. To meet the challenge of the enormous size of the Web, fast, powerful spiders are developed and used to download targeted Web sites into storage tapes.

4- *Web statistics*. The large number of pages collected by spiders is often used to provide useful, interesting statistics about the Web. Such statistics include the total number of servers on the Web, the average size of a HTML document, or the number of URLs that return a 404 (page not found) response. These statistics are useful in many different Web-related research projects and many spiders have been designed primarily for this purpose [14].

4. Structure of a Spider

There are two ways that a spider could be constructed. The first is by writing the spider as a recursive program. The second is by building a non-recursive spider that maintains a list of pages that it must ultimately visit. When we are trying to decide which approach to take, keep in mind that it must allow the spider to function properly with very large websites.

4.1. The Recursive Program

Recursion is the programming technique in which a method calls itself. For some projects, constructing a spider to use recursion seems like a logical choice. It is particularly useful whenever the same basic task must be done repeatedly or when the information for future tasks will be revealed as earlier tasks is processed. For instance, consider the following pseudocode:

```
void RecursiveSpider(String url)
{
 .... download URL ....
```

```
 .... parse URL ....
 for each URL found
  call RecursiveSpider(with found URL)
 end for
 .... process the page just downloaded...
}
```

In this piece of code, the task of looking at one single web page has been placed in a single method called *RecursiveSpider*. Here, the *RecursiveSpider* method is called to visit a URL. Instead, the method calls itself as it discovers links.

Though recursion seems like a logical choice for constructing a spider, it is not a suitable one unless there are relatively few pages to visit. This is because of each iteration must be pushed onto the stack when a recursive program runs. If the recursive program must run many times, the stack can grow very large, which can consume the entire stack memory and prevent the program from running.

Another problem with recursion is encountered when we want to use multithreading, which allows many tasks to run at once. Multithreading is not compatible with recursion because with this process, each thread has its own stack. As the methods called themselves, they would need to use the same stack. This means that a recursive spider could not be extended to include multithreading.

4.2. The Non-Recursive Construction

The second way to construct a spider, and the one that we will be using, is to approach the problem non-recursively. By doing this, we will be writing a spider that uses a method that does not call itself as each new page is found. Instead, this approach uses a Queue and a Hashtable. Figure (1) shows a simple Non-Recursive web spider algorithm.

1. Initialize a page queue with one or a few known sites
   (e.g., http://www.waelsaid.com)
2. Pop an address from the queue.
3. Get the page.
4. Parse the page to find other URLs.
   (e.g., <herf = "http://www.Waelsaid.com/attractions.html">)
5. Discard URLs that do not meet requirements.
   - (e.g., images, executables, postscript, PDF, zipped files, etc.)
   - (e.g., pages that have been seen before)
6. Add the URLs to the queue.
7. If not time to stop, go to step 2.

Figure (1): A Simple Non-Recursive Web Spider Algorithm

5. Web Spiders for Building collections

Use of Web search engines such as Google, AltaVista (www.altavista.com), NorthernLight (www.northernlight.com), Excite (www.excite.com), Lycos (www.lycos.com), and Infoseek (infoseek.go.com) is the most popular way to look for information on the Web. All these search engines rely on spiders to collect Web pages that are then processed to create their underlying search indexes. Examples include AltaVista's Scooter, Google's Googlebot, Lycos's T-Rex, and Excite's Architext.

5.1. Web Spider Architecture

The Web spider traverses the Web starting at a given seed URL. It retrieves HTML files and parses them for new URLs. The URLs are added to a queue to be scanned. Given a domain, the Web spider will crawl over all the links within that domain. One by one, the spider fetches a URL from the queue. Then the Web page is visited and the last modified data of this URL are put into the HashTable. If the content type of the web page is “text/html”, a parser is used to parse the content of this file looking for URLs. After that, the URLs that are not located within the given domain are discarded and new URLs the Web spider parsed out will be checked in the HashTable. If the HashTable does not contain this URL, the new URL will be added to the queue. The Web spider repeats the process above. When there is no URL in the queue, the crawling process is completed. The Web spider system architecture is shown in figure (2).

5.2. Search Strategies

In the study of artificial intelligence, problems can often be reduced to a search. In our Web spider architecture, it will need to utilize a search to navigate through the Web in an effort to reach its goal. Searching is a valuable tool in many applications. It will be a key component in building a Web spider.

Search Strategies can be classified into two basic classes: uninformed and informed. *Uninformed*, or blind, searches are those that have "no information about the number of steps or the path cost from the current state to the goal". These searches include: depth-first, breadth-first, uniform-cost, depth-limiting and iterative deepening search. *Informed*, or heuristic, searches are those that have information about the goal; this information is usually either an estimated path cost to it or estimated number of steps away from it. This information is known as the heuristic. It allows informed searches to perform better than the blind searches and makes them behave in an almost "rational" manner. These searches include: best-first, hill-climbing, beam, A*, and IDA* (iterative deepening A*) searches [15].

Breadth-first search (BFS) is one of the two most common search algorithms every computer science student learns (the other being depth-first search) [16]. The approach of BFS is to start with a queue containing a list of nodes to visit. A node is a state or a value; in programming it is usually represented as a structure containing particular information about the environment or domain of the problem. The algorithm starts by placing the initial state of the problem into the head (beginning) of the queue. The search then proceeds by visiting the first node and adding all nodes connected to that node to the queue. When viewed as a tree graph, it would move from left to right from the current node (usually represented as a circle on a graph) along links (represented as connecting lines in-between the node circles) to connected nodes, adding the connected nodes to the queue. As the head node of the queue is visited it is removed. The search then moves to the next node on the queue and continues until the goal is reached. This search will always find a solution if it exists.

A Web spider that downloads pages in BFS order discovers the highest quality pages during the early stages of the spider process. BFS is a good strategy for Web spider because the most important pages have many links to them from numerous hosts, and those links will be found early, regardless of on which host or page the spider originates [17].

6. Web spiders Implementation

Building a Web spider is a non-trivial endeavor because the data manipulated by that spider is too big to fit entirely in memory, so there are performance issues relating to how to balance the use of disk and memory. We have developed a Web spider using C# language. It's developed using regular expressions, *HttpWebRequest*, *HttpWebResponse*, *webHeaderCollection*, *StreamReader*, *StringBuilder*, *Uri*, *Hashtable*, *sqlConnection*, Threading, and delegates. Snapshots of the C# code are presented below.

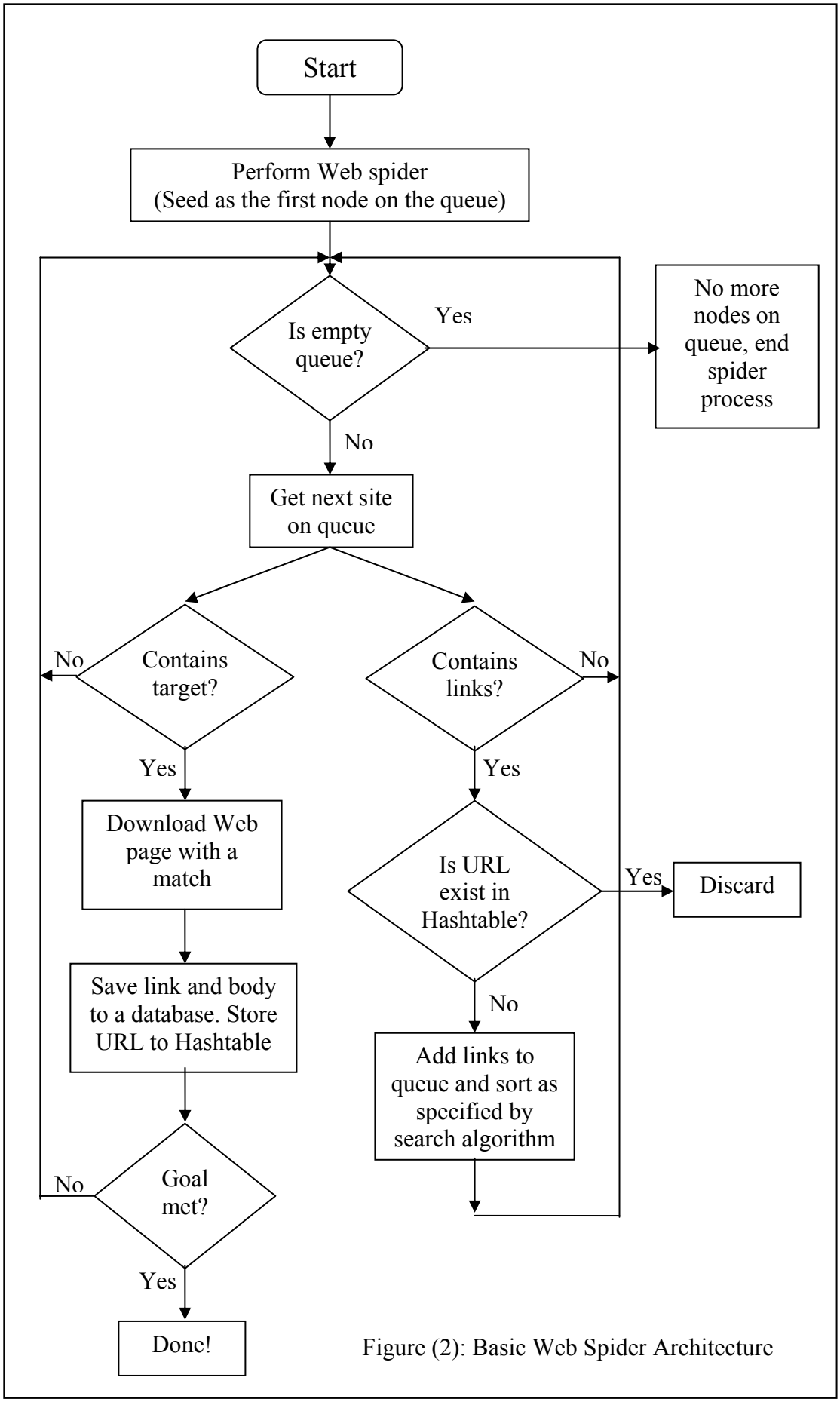

Figure (2): Basic Web Spider Architecture

6.1 Getting web pages as string

Microsoft .NET Framework SDK already provides the necessary classes to fetch a Web page and convert the result into a string. To get a page whose URL is given, we use the *HttpWebRequest* and *HttpWebResponse* classes from *System.Net*. The result is a *StreamReader* that can then be converted into a string using *StringBuilder*.

```
 //Creates an HttpWebRequest for the specified URL .
HttpWebRequest req = (HttpWebRequest) WebRequest.Create (addr);
// Sends the HttpWebRequest and waits for a response.
HttpWebResponse resp = (HttpWebResponse) req.GetResponse();
StreamReader sr = new StreamReader (res.GetResponseStream (), System.Text.Encoding.UTF8);
StringBuilder pageBuffer = new StringBuilder();
String line;
while ((line = sr.ReadLine()) != null )
            {
              pageBuffer.Append(line);
            }
sr.Close();
```

The above is part of the code in the method *loadPage*( ) of the class *webPage*. It is pretty straightforward; a *StreamReader* has a *ReadLine*( ) method, which is used in a loop to append to a *StringBuiledr*.

6.2 Parsing a Web page using regular expressions

After a page is fetched, we have an HTML page in the form of a string. There are many possible ways to parse it [18]. We use regular expressions to collect all the links in it.

```
r  = new Regex("href\\s*=\\s* (?:'(?<1>[^']*)'|\"(?<1>[^\"]*)\"|(?<1>[^>\\s]+))",
RegexOptions.IgnoreCase|regexOptions.Complited);
for(m=r.match(page);m.Success;m=m.NextMatch()
    {
      link = m.Groups(1).ToString().Tolower();
      link = FilterLinks(link);
      if(link.Length!=0) bageOfLinks.Add(link);
    }
```

Although regular expressions look rather cryptic (except to seasoned Perl programmers), they have the advantage of being powerful and efficient. Microsoft .NET regular expressions are compatible to Perl regular expressions, and in addition have an object-oriented outer wrapping. The above classes, properties, and methods (*Match*, *Groups*, *Success*, and *NextMatch*) are used together to process the results of the matching.

7. Conclusions

A Web spider is a specialized agent that follows links as it searches for pages. In this paper, we explored the concept of a spider, we built a spider that could download all of the HTML files for a website, and we learned of the most basic ways of using such a spider. A Web spider is designed to take HTML output from a website and trace its links. By using this process, the spider soon finds other links. This is a recursive operation because a spider is endlessly following these links. The spider in this paper has been built without recursion, however. Recursion would have stack requirements that are too great for a large site. The spider presented in this paper maintains a list of links found. This list is then distributed amount several concurrent threads.

8. Future Work

An important feature of a Web spider is its efficiency. It needs not only to retrieve new URLs and other information correctly, but also to finish the operations as soon as possible. Currently we are still in our middle stage of assembling each piece of code into the

framework and make it run correctly. We will continue our work to optimize the code and accomplish our goal of the efficiency of the Web spider. We will focus our work in the following aspects: code optimization, queue optimization, disk writing optimization, and search engine design.

9. References

[1] Lyman, P. and Varian, H. R. "How Much Information", [Online], Available at http://www.sims.berkeley.edu/howmuch-info/, 2000.

[2] Mitch, M. "Microsoft Internet & Networking Dictionary ", Microsoft press, 2003.

[3] Payne, C. "Building A Web Spider", [Online], Available at http://www.asp101.com/articles/chris/spider/default.asp.

[4] Barry, M. and et al." A Brief History of the Internet ", [Online], Available at http://indigo.ie/~floyd/ih.html.

[5] Chen, H., Chung, Y., and Ramsey, M. "Intelligent Spider for Internet Searching ", In Proceedings of the 30th International Conference on System Sciences (HICSS), Maui, Hawaii , Jan 03 - 06, 1997.

[6] Gray, M. "Internet Growth and Statistics: Credits and Background", [Online], Available at http://www.mit.edu/people/mkgray/net/background.html, 1993.

[7] Chau, M. and Chen, H. "Comparison of Three Vertical Search Spiders", IEEE Computer, 36(5), 56-62, 2003.

[8] Chau, M., Zeng, D., and Chen, H. "Personalized Spiders for Web Search and Analysis", In Proceedings of the 1st ACM-IEEE Joint Conference on Digital Libraries, Roanoke, Virginia, USA, Jun 2001, pp. 79-87.

[9] Ebada, A., Iraki, H., and Wael, S. "A New Model for C# Client-Side Search Agent", in Proceeding of the 12th International Conference on Artificial Intelligence Applications (ICAIA'04), Cairo, Egypt, Feb 18-20, 2004.

[10] Heydon, A. and Najork, M. "Performance Limitations of the Java Core Libraries", In Proceedings of the 1999 ACM Java Grande Conference, Jun 1999, pp. 35-41.

[11] Bharat, K. and Broder, A. "A Technique for Measuring the Relative Size and Overlap of Public Web Search Engines", In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, Apr 1998.

[12] Henzinger, M. R., Heydon, A., Mitzenmacher, M., and Najork, M. "On Near-uniform URL Sampling", In Proceedings of the 9th International World Wide Web Conference, Amsterdam, Netherlands, May 2000.

[13] Kahle, B. "Preserving the Internet", Scientific America, Mar 1997.

[14] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. "Graph Structure in the Web", In Proceedings of the 9th International World Wide Web Conference, Amsterdam, Netherlands, May 2000.

[15] Russell, S., and Peter, N. "Artificial Intelligence A Modern Approach", Prentice Hall, Upper Saddle River, N.J., 1995.

[16] Thomas, H., Charles, E., and Ronald L. "Introduction to Algorithms", McGraw-Hill, New York, N.Y., 1997.

[17] Najork, M. and Wiener, J. L. "Breadth-first Search Crawling Yields High-quality Pages", In Proceedings of the 10th International World Wide Web Conference, Hong Kong, May 2001.

[18] Cho, J. and Sridhar, R. "A Fast Expression Indexing Engine", *In Proceedings of the Workshop on Management of Semistructured Data* held in conjunction with ACM SIGMOD'97, pages 18-25, May 1997.