JOB AGGREGATION SEARCH ENGINE

A Paper
Submitted to the Graduate Faculty
Of the
North Dakota State University
of Agriculture and Applied Science

By

Anita Sundaram

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

February 2011
Fargo, North Dakota

North Dakota State University
Graduate School

Title

JOB AGGREGATION

SEARCH ENGINE

By

ANITA SUNDARAM

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Sundaram, Anita, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, February 2011. Job Aggregation Search Engine. Major Professor: Dr. Kendall E. Nygard.

In this paper we describe the design and implementation of a Job Aggregation Search Engine (JASE) that acts as a one-stop-shop for listing recently posted jobs across top multiple job search engines. There are multiple job search engines available that receive and present jobs posted by employers. The JASE system extracts data from multiple websites and presents the job data in a consistent and presentable format.

The objective of this paper is to implement a job search tool that seeks to reduce the browsing time of the user querying multiple job websites for the same job criteria. It also aims to reduce the possibility of the user being overwhelmed while browsing through various websites to find the job of interest. The reduction in total browsing time is made possible by triggering a search for jobs when the user chooses a discipline. This allows the user to view recently posted jobs across multiple job boards. Often it can become tedious for a user to visit a job website and not find the job of interest, resulting in browsing through other websites one at a time. In order to avoid the browsing of many sites, JASE serves as a job extraction program that aggregates jobs from multiple job websites and returns results in a simple, user-friendly user interface.

The program consists of two components, the user interface and the job extraction program. The job extraction program has two components, namely the crawler program and the parser program. The techniques for crawling and parsing the websites are designed and implemented after carefully studying the HTML structure of the target website.

ACKNOWLEDGEMENTS

I would like to thank Dr. Kendall E. Nygard for his continued support, help and direction. My sincere thanks to Dr. Changhui Yan, Dr. Tariq King, and Dr. Limin Zhang for serving on the committee. I would also like to thank my parents who gave me encouragement to complete the paper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

1. INTRODUCTION

In recent times, Job Search Engines have become an important bridge between job seekers and employers. These engines provide job seekers with access to job vacancies posted by companies and also, an easy way to apply for the jobs online. One of the underlying concepts used is that of a metasearch engine or a vertical search engine or a combination of both. A Metasearch engine [1] crawls through various search engines and presents the output without the intermittent usage of a database; whereas, a vertical search engine [3] aims at scraping data that pertains to a pre-defined topic such as jobs and careers.

The primary goal of a job search engine is to provide a listing of all available jobs in the Internet by retrieving job vacancies from career webpage of multiple company websites in addition to job vacancies posted to the job engine website by individual companies.

There is a large amount of job data in the Internet, which becomes a target for many search engine enthusiasts. Each of the search engines employs their own search algorithm techniques to extract available information. Due to the variance in the crawling techniques, not all engines crawl through the same web pages. Hence, when a user queries different search engines for a word or a set of words, he may find different results being returned.

The objective of this paper is to reduce the browsing time of the user from visiting various websites for the same job criteria a user is interested in and also to reduce the overwhelming factor associated with browsing through multiple job search engines for relevant jobs.

This paper implements a Job Aggregation Search Engine(JASE) which aims at categorizing jobs into sectors such as Accounting, Computers, etc; retrieving job listings

from top job portals in real-time; unifies the listings and gives a response in a consistent format so that users can choose the best website that suits their job interests. The implementation consists of a user interface which displays a list of job categories. Once the user selects a category, the crawler program is triggered. It crawls through various top job websites and displays a unified list of related jobs that are posted on the current date. This method of unification saves the user from accessing various sites and keying in the same search data multiple times.

The two main components related to job extraction are the crawler and parser program. In general, a web crawler is a computer program that browses through the internet in a methodical manner. The crawler program visits the list of seed job URLs, and downloads all the required child hyperlinks such as the job listing hyperlink. The parser program is used for text processing where in it parses through the downloaded HTML content of the job listing page and extracts only the desired text and hyperlinks such as Job Position name, Location, and Description.

Each job website has a different layout and HTML structure. The extraction technique involves analyzing the HTML content and employing different crawling and parsing programs to extract only the target data. An attempt has been made to standardize the crawlers and parsers to an extent such little modification is required for different websites.

The rest of this paper is organized as follows: Chapter 2 discusses about earlier work that has been done with job search engines. Chapter 3 describes the architecture of JASE. Chapter 4 discusses about the crawling and parsing techniques employed. Chapter 5 and Chapter 6 illustrate the output and testing results respectively.

2. RELATED WORK

Due to the increase in the amount of data present on the web, research is being done on search engine techniques to provide a unified access to multiple search engines and consolidate the results into a single list. This section discusses work that is related to this paper and about search engines that has been used as a target source for the search tool.

Metasearch engines are systems that take requests from the user and sends them to multiple other search engines or databases and unifies the results into a single list, thereby, saving the user from visiting multiple websites separately [1]. The main advantage of these engines is that they provide larger coverage and a consistent interface [2]. These engines do not use a physical database to store the results. Rather, the databases of the search engines are used to collect the results and display it in a list format. These metasearch engines employ unique search algorithms to implement the search. The architecture of a metasearch engine [3] is illustrated in Figure 1.1.

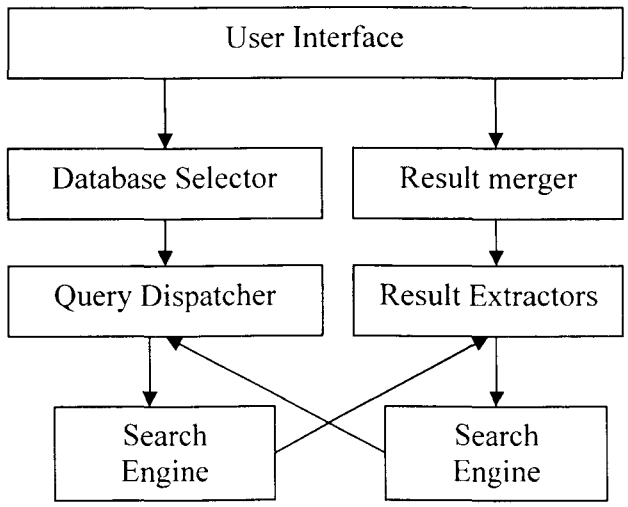


Figure 1.1. Architecture of Metasearch Engine [3]

The metasearch engine [3] takes the input from the user using the User Interface, selects the correct database, customizes the query and sends it to multiple search engines. The result from these search engines are merged together and returned to the user. This is similar to our work because the user query is sent to multiple search engines and the results are merged into a single unified list. Since we restrict our search to the job domain, the user query is sent to all the selected search engines without any database selection criteria.

There are many Job portals that exist as typical search engines which aggregate jobs from job websites, career websites and other online sources. A metasearch technique is used in paper [4]. It discusses the a job search techniques where in it extracts the user interface from job boards, generates an XML scheme, and uses resultant information to display jobs. Our result is related to this work. However, we extract job information from the search engines job webpages based on the HTML source code.

Vertical Search Engines [3] use focused crawlers. Instead of crawling the entire web; search is based on a certain domain. This work involves text and link analysis for the crawling. Our paper is related to this work since we focus on only the job sector. Hence, text in the HTML webpage is analyzed to crawl and parse the job information.

Indeed.com [5] and Simplyhired.com [6] are metasearch engines that browse through thousands of job boards, classified listings, newspaper etc. These engines focus on a keyword search. These engines do not have a physical database but uses the databases of the source websites. Our paper is related to the work since we also crawl through these websites to gets gather result sets. But in our case, a database is used for intermittent storage of extracted job information from where the results are displayed. Also, our focus is on a

job sector categorization. The result is displayed on a single page with important job data such as job title, location and descriptions link.

Job.com [7] is a job search engine that provides an extra option for the users to post their resumes and employers to post their job listings. The paper is related to this work but we provide the users with a job sector selection drop down menu as the main source of input. We crawl through job.com and display the results in a tabular format which is easily comprehensible. The user can click on the hot-links provided on the display page to apply for jobs through job.com. JASE holds a database from where information is manipulated to a simple format. Job information from all these job websites are merged together to provide a consistent result.

3. COMPONENTS OF JASE

The structure of the job engine can be classified into two segments. The first segment is comprised of a user interface and the second segment is the data extraction program. The user interface is the web based medium through which the user inputs the job selection criteria and gets the results displayed in a readable format. The job extraction program mines through various job resources and parses specific job information on those job board websites and downloads job data into the database. Section 3.1 briefs about the environment used to build the system. Section 3.2 discusses about the user interface and section 3.3 discusses about the data extraction program which includes the crawler and parser programs. Figure 3.1 illustrates the architecture of JASE.

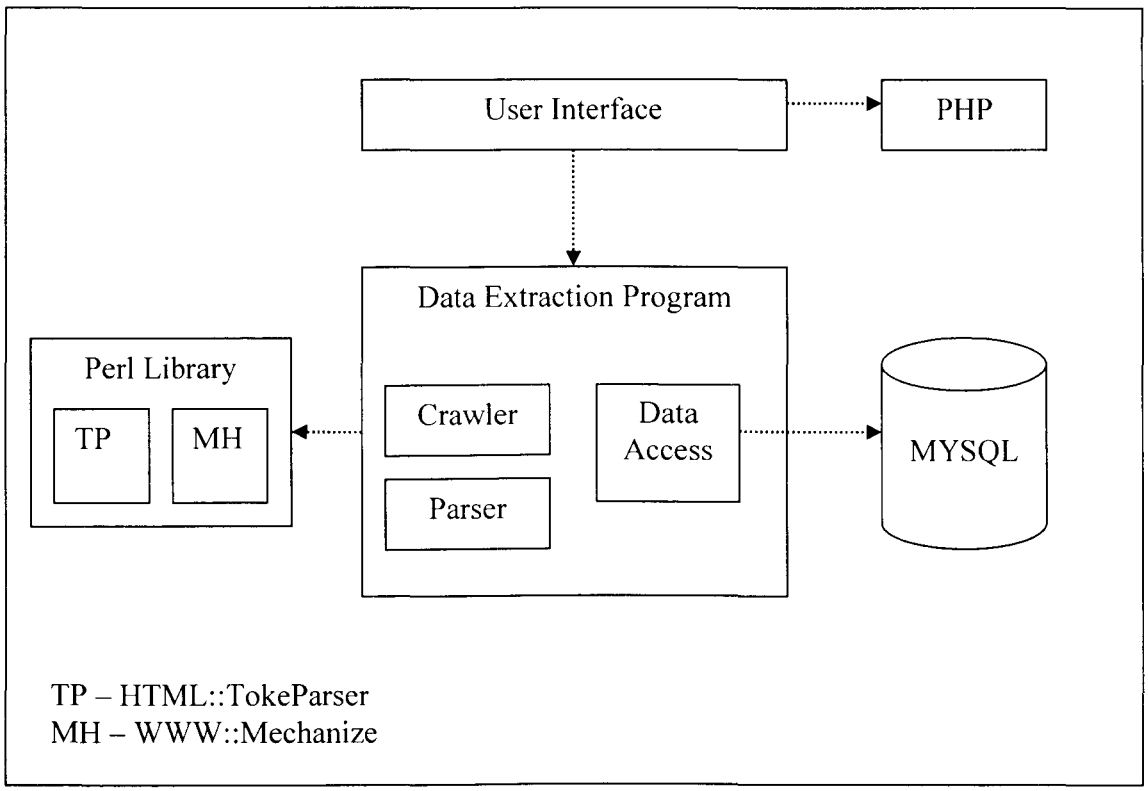


Figure 3.1. Architecture of JASE

3.1. Environment Used to Build the Search Engine

The environment used to develop the user interface is XAMPP [8]. It is an open source, cross platform development tool used for executing Perl and PHP programs using Apache server and MySQL database. The main reason for choosing XAMPP [8] environment is that it can be run on Windows and it is easy to configure to make it a web server. The user interface was built using PHP, HTML and JavaScript. PHP is a free software and scripting language used to generated dynamic web pages. It is used for retrieving data from database and presenting in a web based format. Perl is used as the data extraction language. Perl is a high level, general purpose programming language that has powerful text manipulation facilities. Perl is chosen because the underlying technique for the data extraction program is based on text identification and extraction from websites.

3.2. The User Interface

The User Interface provides the user with a dropdown menu that lists various job sectors. Interface is intended to be simple and easy. Categorizing of jobs is used as the criterion for job retrieval because users can easily relate to a job domain depending on their profession. For instance, an accountant can select Accounting as the sector and browse through all related jobs. Figure 3.2 shows the job selection input page.

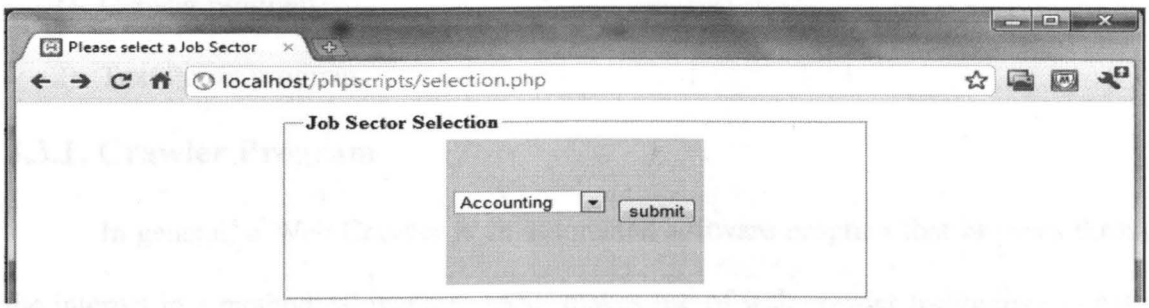


Figure 3.2. Screenshot of the User Interface Showing the Job Sector

Once the user selects the sector from the drop down menu and hits the submit button, he will be routed to a page that displays a unified list of jobs from various top job portals. The job listing page displays details in a tabular format which is easily comprehensible. Details that are listed are title of the job, location where the position is available and the details hyperlink that route to the web page where it is originally posted. The details hyperlink redirects the users to the appropriate detailed description page wherein he usually has an option of applying for a requirement online.

Job data available in the job boards are extracted after the users submit action. Real-time extraction of data has been done giving the users an advantage of browsing through the latest jobs. In order to limit users' wait time, a partial list of jobs is displayed periodically as and when a particular job site is extracted. The user can browse through the jobs that are initially displayed while the rest of jobs get crawled and loaded.

Once all the jobs are loaded, the users are given an option of sorting out the job title and location in ascending or descending order.

3.3. Data Extraction Program

The core part of the JASE lies in data extraction. The data extraction segment of the job retrieval program can be divided into two units.

1) Crawler program
2) Text Parser program

3.3.1. Crawler Program

In general, a Web Crawler is an automated software program that browses through the internet in a methodical manner. JASE makes use of web crawler technology to gather

information from a given set of publicly available large job boards and retrieves relevant and up-to-date data.

Once the extraction system receives the job sector input from the user, it triggers the crawler program. The crawler is fed with a seed URL which is the entry point to the job website that has to crawled. The seed URL is not always the main page of the job website. Depending on the job sector, the seed input keeps changing. The techniques employed for crawling will be discussed further in chapter 4.

The web crawler automatically navigates through the URL and downloads the destination page as HTML content in its history stack. All the pages that are fetched can be traversed back and forth. Due to the large number of jobs in job boards, the websites have pagination facility. Hence, the crawler loops through all the pages until a specified break condition. The downloaded page will be used by the parser program for text processing as discussed in section 3.3.2.

One of the important behaviors of the web crawler is related to the Politeness policy [12] that states how to avoid overloading websites. Steps have been taken while building the crawler to specify the number of seconds delay between each request for next page in order to avoid overloading the other job websites servers. Some of the modules that are employed in writing the program have been discussed in the following sections.

3.3.1.1. Mechanize

Mechanize [10] is a Perl module that has been used for the process of web crawling. It is a handy module that aids in programmatic web browsing. It is used to automate interaction with websites. It is used in the process of crawling and screen scrapping which not only downloads hyperlinks but also emulates an interaction with the website facilitating

navigation around site and filling out forms. Mechanize can be used to fetch sequence of pages by following links and submitting forms. The visited URLs can be queried and revisited since mechanize stores the history of all the URLs.

Reasons for choosing Mechanize are as follows:

1. Job search engine provides a feature keyword search box and a submit button. Mechanize provides methods for automated form filling and submitting forms.
2. Each fetched page by Mechanize is stored in history. So, traverse between pages becomes easy. It mimics a back and forward button in a web browser.
3. It is used for testing web applications. Using the testb::* modules, the fetched content can be checked and passed as an input to a test call.
4. It automatically handles cookies and redirections.

One disadvantage of Mechanize is that it cannot be used to crawl JavaScript pages.

Declaration and Methods used with Mechanize are described below:

A Mechanize object can be created using the following syntax:

```
my $mech_object = WWW::Mechanize->new().
```

Some of the methods that are commonly used to crawl web pages are discussed as follows:

1. $mech->get($URL)

   This method is used to the fetch the job URL that is passed as a parameter. The URL can be passed as a string or URI object or a mechanize link.

```
My $link = www::mechanize::link->new({url => $url,
                                      Text=> $text,
                                      Name => $name,
                                      Tag => $tag});
```

2. $mech -> back()

   This method is used to return to return to the previous webpage. It is similar to the back button of a web browser.

3. $mech -> links()

   It is used to get a list of all the links found on the last fetched page.

3.3.2. Parser Program

The downloaded webpage received from the crawler program is the input to the parser program. A webpage is generally built using HTML. An HTML page consists of a number of tags which encloses text, hyperlinks and sub-tags. The parser program parses through the HTML content of the web pages to extract job data.

As discussed, the important details that are required for a job listing webpage are the job title and corresponding location and the description link. The main intent of the parser is to scrape only these details avoiding all other text content on the webpage. Figure 3.3 shows a sample layout of a job listing page of a search engine.

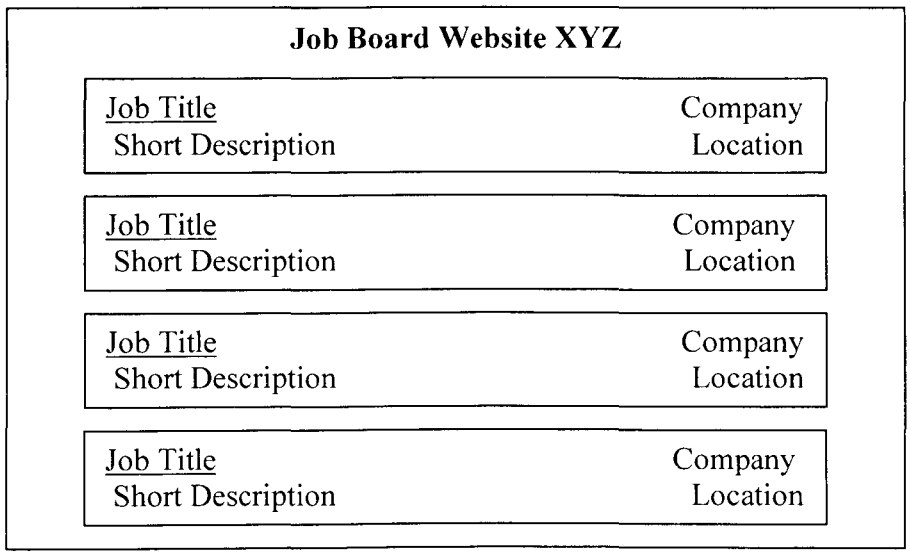


Figure 3.3. Sample Layout of Job Listing Page of a Job Search Engine

A job board website generally follows the same HTML structure for its job listing pages. The job postings along with its short description would be displayed vertically one below the other and the corresponding details such as location are located at specific positions. In Figure 3.3, all the individual job titles would be placed between the same start and end HTML tag. The parser would search for the job title start tag and then looks for the tag which contains the corresponding Company information and Location. These HTML start tags will sometimes be accompanied by attributes such as href, class, id, style or title. In such cases, a combination of start tag and attribute will be used by the parser to identify and extract the required text information. The HTML source for Figure 3.3 is shown is Figure 3.4 below.

```
<div style = "..."> Company</div>
<h2 class = "job_title"><a href ="/job_title_link/">Job Title</a></h2>
<div>Location</div>
```

Figure 3.4. HTML Source for Individual Job Listing in Figure 3.3

In Figure 3.4, the parser can parse the HTML document for the first div tag and style combination to find the company detail. It would find the heading tag h2 with attribute class that equals job_title, in order to get the job title information. It finds the anchor tag with href to extract the details link and it follows the second div tag to get the location.

The extracted text information is stored in a database. These details are retrieved from the database using PHP language and presented in a simple, readable format. Parsing techniques will further discussed in Chapter 4.

For testing purposes, the parser program is used to concatenate and return an output of the extracted text information in a readable format i.e., in the form of a notepad or an excel file. This can be done before sending the information to the database.

The Perl modules and methods that are used for data scraping are discussed in section 3.3.2.1.

3.3.2.1. HTML::TokeParser

HTML::TokeParser [11] is a Perl module that is used for text extraction. It is a class with predefined set of token types and it has methods that identify tokens.

It is initialized as follows:

```
$parse_content = HTML::TokeParser->new($file, %opt )
```

where $file can be a file name, plain scalar that is an entire document that has to be parsed.

3.3.2.1.1. Advantages of TokeParser

TokeParser provides efficient methods to extract the required data such as URLs, title of the URL link, any text on the HTML page given a opening tag and an closing tag.

It provides methods for rewriting start tag attributes. It is used when there is incorrect HTML on the page which in general cases, neglected by the parser.

3.3.2.1.2. Important Methods used in TokeParser

1) get_tag()

   The method is used to return an HTML tag. Since relevant job data on the job webpage will be enclosed within an HTML start tag and an end tag, the first step for text extraction in our case would be to get the tags.

```
$tag = $stream->get_tag('HTML tag')
```

2) get_text()

The get_text method would return all the text that is available in the current position of the tag or till the specified tags that passed as arguments to the method.

$text = $stream->get_text('HTML tag')

4. IMPLEMENTATION OF DATA EXTRACTION

The information retrieval combines the crawler and the parser program which aids in information retrieval from large job boards. Each job search engine has unique HTML structure that is analyzed and an appropriate crawler and parser is written. These crawler and parser programs have a standard format except pieces of code that are modified due to the difference in the main seed URL and HTML tags that contain the core job details.

The prototype job retrieval program extracts job information from top five job search engines. These search engines have jobs that are either posted to their websites by companies or jobs that are crawled through various companies career website or both.

4.1. Strategy for Crawler Program

To initiate the crawling of a website, the crawler is given an input of the target job search engines URL. Since the categorization of jobs is done based on the job sector, an initial analysis of the website is done to find the URL that directly routes to the jobs listing page of a particular URL. Taking the given link below as an example, the analysis can be explained.

```
$URL = http://www.simplyhired.com/a/jobs/list/q-accounting/fdb-1/pn-1
```

The main link of a popular job search engine is www.simplyhired.com [6]. The first step is to find the link that contains the accounting jobs. The parameter "q-accounting" specifies that the accounting sector is taken into consideration. Once the link has been clicked, it routes to the first page of the job listings. In order to crawl only the latest jobs, a

criterion of data posted is taken into account. In the case of simplyhired.com website, the latest jobs are posted using a separate folder name "fdb-1" in addition to the existing accounting URL. Hence this complete link is given as a seed URL to the crawler program.

Similar techniques have been followed for crawling other job search engines seed URL. In order to crawl only the fresh jobs, the job posted date on the website has to be taken into account. There will not be an exclusive hyperlink or a hot-link for all the categories. Hence, further analysis has been done to gather the appropriate seed URL.

The crawler program clicks on the seed URL and downloads the HTML content of the webpage. These sector wise jobs are displayed in multiple pages using pagination. In general cases, about ten pages are listed per page. The parameter "pn-1" in the URL represents the first page of the jobs sector webpage. The crawler then crawls through all these webpages and these corresponding hot-links are stored in its history stack.

The technique used to crawl and extract job information from a famous website has been discussed in section 4.2.

4.1.1. Crawling Sector Based Seed URL and Related URLs

Seed URL is decided in such a way that the crawler need not travel through multiple links to reach the sector based jobs. This contributes to the reduction in the total time of crawling. On analyzing the seed URL, it is found that there exists a parameter that can be incremented to travel through multiple pages. The crawler loops through the given'nextpage' URL and downloads all the pages. On a generic level, the input to the crawler program can be divided into common categories.

- Job search engines have a default search box where the user can input the job sector name. Perl module Mechanize is used to mimic the user input. The search box and the submit button are generally the placed inside an HTML 'form' tag.

```
@array = {'Accounting', 'Administrative', 'Finance'};
Foreach $sector(@array)
{
        $sector = $array;
        $url = "http://xyz.com";
        $browser -> get($url);
        $browser -> form_number(n);
        $browser -> field('search_string',$sector);
        eval{
        $browser -> click_button(value => "Search");
        };
$contents_of_webpage = $browser->content();
}
```

In the code given above, the job sectors are stored in an array. The crawler looks for the form number where the search box is located and auto fills the job sector and hits the Search button. The HTML content is then stored in a variable.

- In some cases, the job sectors are listed in a Drop-Down menu. On analysis of the source code for HTML, the form encloses the Drop-Down menu and the

submit/search button. In the HTML source code, the crawler looks for the option value. A sample source code is shown below:

```
<form action ="/www.xyz.com" method = post>
        <strong class = "abc">Job Category:</strong><br/>
        <select name = "industry" id = "xyz">
                <option value = "1">Accounting</option>
                <option value = "2">Administrative</option>
                <option value = "3">Finance</option>
        </select>
</form>
```

Taking into consideration the code above, the strategy for crawling follows the one employed for the search box style crawling with the exception that it uses the name of the drop down menu along with the option value.

```
$browser->value ($name, $number)
Where $name = industry
$number IN (1, 2, 3)
```

The webpage that is crawled by the crawler is used by the parser program which in turn parses through the source HTML to extract the job information.

4.2. Strategy for Parser Program

Parser program is implemented in such a way that it looks for a specific set of HTML tags in the HTML source code. Search engine websites are generally designed in an organized manner. Individual job titles are listed in a webpage in systematic manner. There is usually a unique tag that encloses the job titles. The job titles also work as a hot-link which routes to the detailed description page of a job source website's page. Sample HTML layout of a job website is illustrated in Figure 4.1.

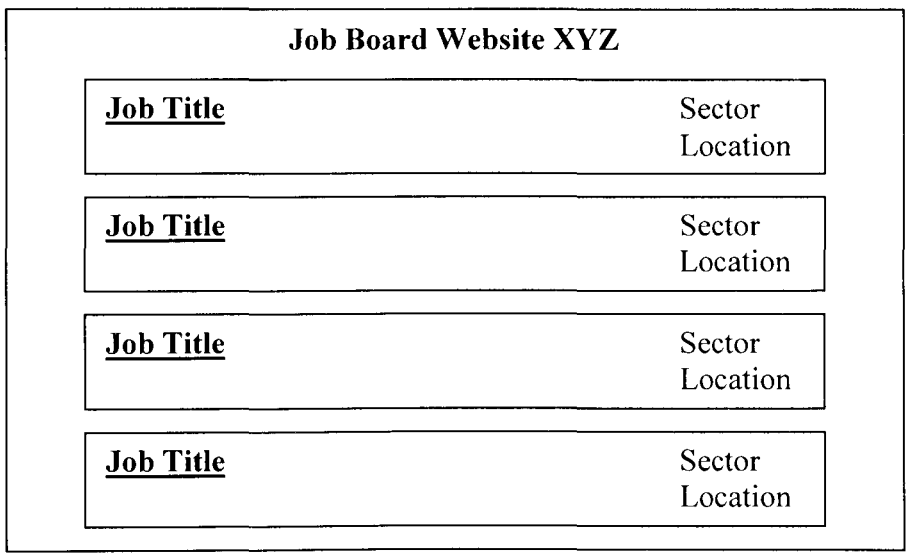


Figure 4.1. Sample HTML Layout for Job Listing Page

In figure 4.1, it can be seen that the Job titles are located vertically one below the other. The corresponding job sector and location are positioned on the right of the page relative to the job titles. The HTML source code of the sample page in Figure 4.1 is shown in Figure 4.2.

```
<h2 class = "jobtitle_result">
        <a id = "abc1" href = "/job/a=1/b=2/" >Job Title 1</a>
</h2>
<div style = "float"right"> Location 1</div>
. .
. .
. .
<h2 class = "jobtitle_result">
        <a id = "abc2" href = "/job/a=1/b=3/" >Job Title 2</a>
</h2>
<div style = "float"right"> Location 2</div>
```

Figure 4.2. Sample Source Code for Figure 4.1

On analyzing the HTML code in Figure 4.2, it can be seen that the individual job listings are placed inside heading "<h2>" tags with an attribute class equaling "jobtitle_result". It should be made sure that this combination is applicable only to the job titles in the entire HTML source. If it holds good for other information apart from the job details, the parser would be extracting irrelevant information.

The parser would initially get the h2 tag and then looks for the element "<a id =" followed "href" which holds the name of the job title as well the hot-link for detailed description.

The immediate division tag " " is parsed for the location of the job. Similarly, the company details, date posted can be also be extracted by looking into the correct HTML tags.

4.3. Data Retrieval from a Target Job Website

Indeed is a large job search engine that crawls through thousands of jobs on the internet. The data retrieval technique combines the crawler and the parser programs. Once the crawler accesses the input URL of indeed.com [5], the parser scrapes the job data.

4.3.1. Crawler Program for Seed URL

The crawler program crawls jobs related to job categories such as Accounting, Administrative, Computer and Finance. The URL of the source website contains the category name. The names of the categories are stored in Perl associative arrays:

$category {'number'}{'name'}

where a random number is assigned and "name" equals the category name. The code snippet the job selection is shown in Figure 4.3.

```
#!/usr/bin/perl
use strict; # implementing strict helps in writing quality and cleaner code
use WWW::Mechanize;
use HTML::TokeParser;
my $bot_name = 'Indeed';
my %sectors; # associative array which would contain the different
disciplines for a job seeker
# Sector selection
$category{'1'}{'name'} = 'Accounting';
$category{'1'}{'sector_id'} = '1';
$category{'2'}{'name'} = 'Administrative';
$category{'2'}{'sector_id'} = '2';
$category{'3'}{'name'} =  'Finance';
```

Figure 4.3. Code Snippet for Job Sector Selection Page

The code snippet for crawling the seed URL is shown Figure 4.4.

```
        my $nextpage = 'http://www.indeed.com/';
        # Initialise a new Mechanize instance for the main page function, and the
        subpage function
        my $page = WWW::Mechanize->new();
        print "\nConnected to Indeed..... \n";
        #Indicates the current category we are scraping
        print "\nScraping \"".$category{1}{'name'}."\"\n\n";
        my %jobs;
        my $job_ref;
        my $jobcounter = 1;
        my $counter = 0;
        my $counter1 = 1;
        my $numpages = 0;
        my $total_pages = 0;     #it is not the total number of page but the existing
        page.
        my $detailspage;
$nextpage =
'http://www.indeed.com/jobs?q='.$category{$admin_category}{'name'}.'&start=0'
;
$page->get($nextpage);
my $cleancontent =  $page->content;
```

Figure 4.4. Code Snippet for Crawling Seed URL

"http://www.indeed.com/jobs?q=Administrative&sort=date&start=0" contains the first page which has job in the accounting discipline. Before we begin browsing through the website, we need to get the total number of pages on the target website which has Accounting jobs. This section identifies the total number of pages to be crawled and text parsed. We step through HTML on the target website and scrape the total number of jobs

actually found for the current discipline search. This is then divided with the total number of jobs usually found on the target website to get the total number of pages to be scraped.

4.3.2. Crawler Program for Multiple Pages

The job search engines have the jobs categorized by sectors that are spread across multiple pages. Hence, the crawler program has to recognize the link for the next pages. The websites display the total number of jobs for a particular sector. This display is utilized by the crawler program to crawl through multiple pages. Parsing techniques are used by the crawler program to find the total number of pages which is illustrated in Figure 4.5 and 4.6.

```
my $stream = new HTML::TokeParser(\$cleancontent);
while(my $tag = $stream->get_tag('td'))
    {
        my ($temp, $first, $second) = '';
        if($tag->[1]{class} eq "search_meta")
        {
            $total_pages = $stream->get_trimmed_text('/b');
            ($temp,$total_pages) = split('of ', $total_pages);
            ($temp1, $temp2) = split(',', $total_pages);
            $total_pages = $temp1.$temp2;
              if($total_pages%10 == 0)
              {
                 $total_pages = $total_pages/10;
                 print "Total Pages = $total_pages\n";
              }
              else
              {
                 $total_pages = $total_pages/10;
                $total_pages = int $total_pages;

                $total_pages = $total_pages + 1;
                print "Total Pages = $total_pages\n";

              }
              print "\n";
              last; }}
```

Figure 4.5. Code Snippet for Finding Total Number of Pages for a Job Sector

```
for ($l=1;$l<$total_pages;$l++)
        {
                $nextpage=
                'http://www.indeed.com/jobs?q='.$category{$admin_category}{'nam
                e'}.'
                &start='.$counter;
                $page->get($next page);
        }
```

Figure 4.6. Code Snippet to Crawl Multiple Pages for a Specific Job Sector

4.3.3. Parser Program for Text Extraction

This initial scraping actually gets to the piece of HTML code which actually has the title of the job. This is accomplished by getting to the HTML tag that contains the title, comparing it against a condition that is unique in the entire source HTML page and also checking if the condition applies to all the job titles on the page. After this determination, we specify the condition and title is scraped with the 'get_trimmed_text' function in the HTML::TokeParser to remove unwanted white spaces in the job title.

After the title of the job is extracted, the href attribute of the anchor tag needs to extract to scrape the link to the full summary for the current job. This is done using the $tag->[1]{href} function. There are two kinds of URL, some websites have the absolute URL for the detailspage and some have the relative URL. If the website has a relative URL, we would concatenate it with the site URL to form an absolute URL to the detailspage. The location of the job and job post date can also be scrapped depending the website and would add to the list of key information that is required to identify the job.

In order to perform the first level of de-duplication, we would need to create a primary key for the job to make sure they are not scrapped again when we run the program again. The absolute link to the details page carries a reference number for the job or a job ID

which can also be scrapped using regular expressions (REGEX). This can act as the primary key for the job so a comparison can be made against these ids's to determine if we already have these jobs scrapped. This eliminates listing duplicate job and would enhance user experience. This technique of information retrieval is applied to every job on the current page and to every page on the target website until we get the last page that we calculated initially. This ensures a large pool of jobs that a user can apply. The above technique is found to be appropriate for a wide variety of job boards that have an organized listing of job on their site. The code snippet for parsing a target website is shown in Figure 4.7.

```
my $cleancontent =  $page->content;

# Initialise a new TokeParser instance
        my $stream = new HTML::TokeParser(\$cleancontent);
        while(my $tag = $stream->get_tag('h2'))
        {
           if($tag->[1]{class} eq "jobtitle")
           {
#scraping links here:
                $tag = $stream->get_tag('a');

                $detailspage = $tag->[1]{href}; # /job/337095/head-of-it-services?vsrc=1
       #NOTE: scraping position of the job:
                $jobs{$job_ref}{'name'} = $stream->get_trimmed_text('/a');

                $jobs{$job_ref}{'link'} = $site.$detailspage;

                $jobcounter++;
                print "\n";
                print "\t";
                print $jobs{$job_ref}{'name'}."---".$jobs{$job_ref}{'link'}."\n";

            }
         }
# Wait between 3 and 5 seconds before getting the next page.
sleep int(rand(5 - 3)) + 3;
```

Figure 4.7. Code Snippet for Parsing Target Website

5. OUTPUT

The user selects the input sector which triggers the crawler program and a list of jobs from five different job board engines are returned. The crawler takes into account some of the famous websites such as indeed.com [5], simplyhired.com [6] and job.com [7]. As crawling of the websites is done real-time, the job results from one website are displayed initially so that the user can browse through it and apply for the interested job. It crawls through the websites one by one and displays the results in a web browser. In Figure 5.1, the job sector named Administrative has been chosen.

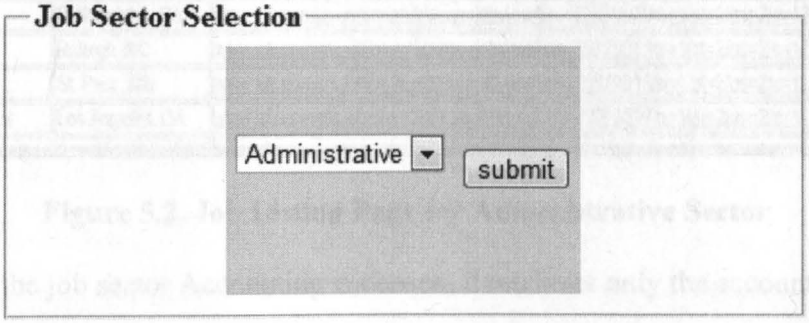


Figure 5.1. Selection of Job Sector - Administrative

When the submit button is clicked, the crawler retrieves jobs from the website called job.com [7]. It displays the job title, location of the job and the details link which is a hyperlink that routes to the description page of the main source website. It then retrieves the jobs from a website called indeed.com [5] followed by simphired.com.

Initially the extracted jobs details from job.com [7] are displayed in a tabular format. After a delay of a few seconds, jobs from indeed.com [5] and simplyhired.com [6] are retrieved. Once all the jobs are retrieved, the user is given an option of sorting by job title and location. The sorting option is more presentable because it gives the user clarity of the

location of the job he is interested in. Figure 5.2 displays the results for the sector Administrative.

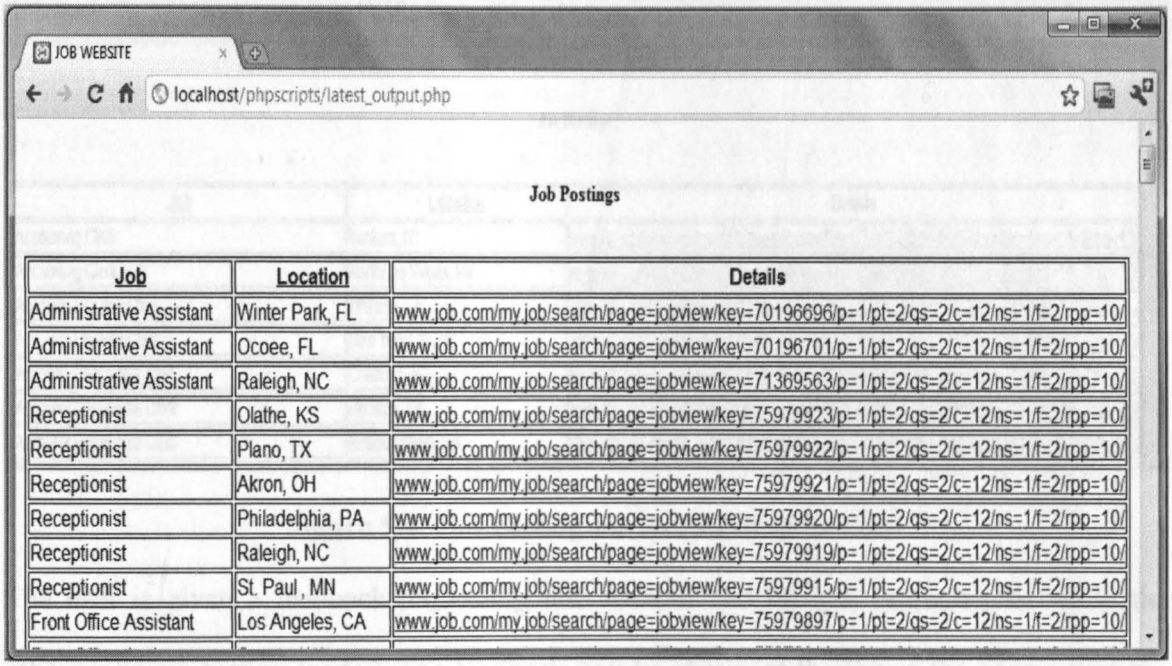


Figure 5.2. Job Listing Page for Administrative Sector

When the job sector Accounting is chosen, it retrieves only the accounting jobs from the given search engines but initially displays job from the website called www.job.com as discussed earlier. The Accounting sector is chosen in Figure 5.3.

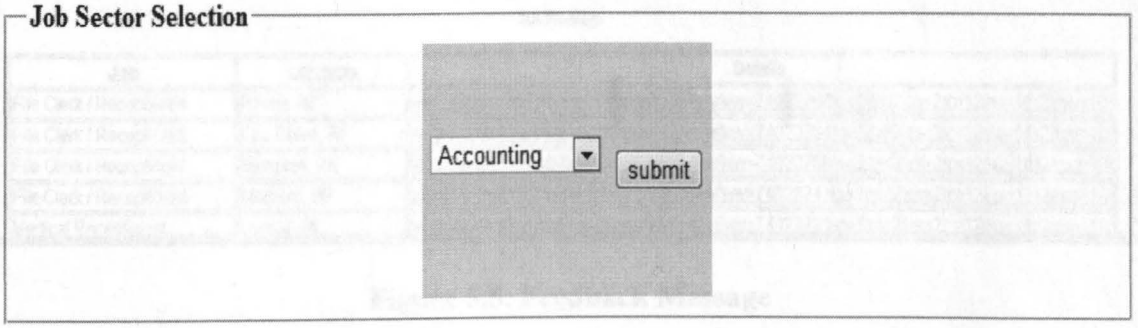


Figure 5.3. Selection for Job Sector - Accounting

The corresponding screenshot of the accounting job listing is shown in Figure 5.4.

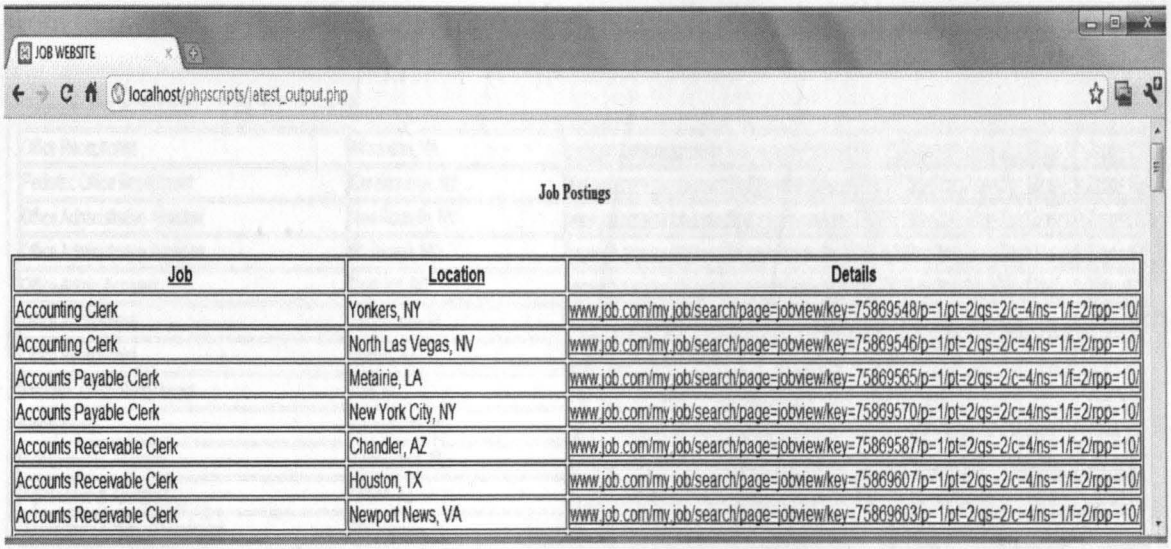


Figure 5.4. Job Listing for Accounting Sector

The user is given a feedback requesting him to wait and browse through jobs while the others jobs are getting loaded. The feedback will be shown at the top of the page above the job listing table. Illustration of the feedback is show in Figure 5.5.

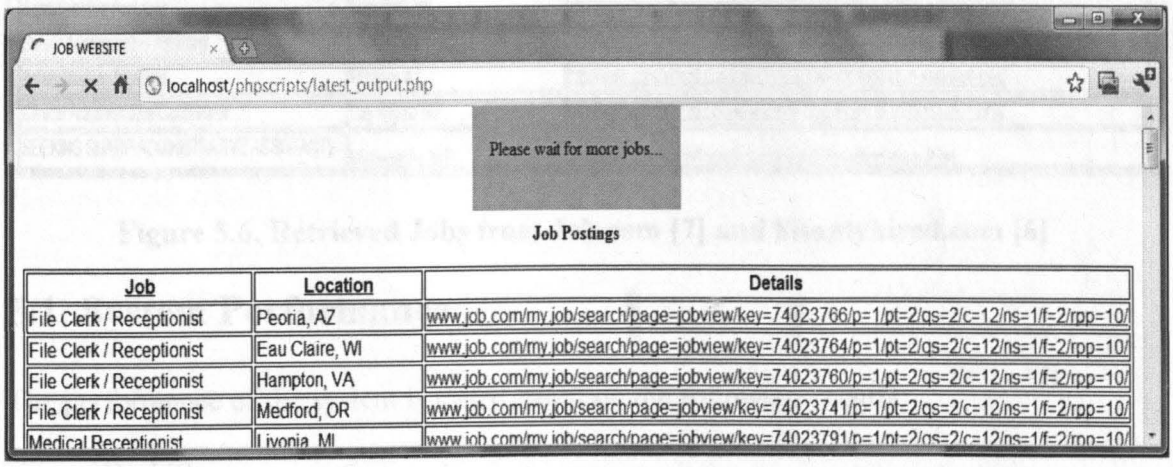
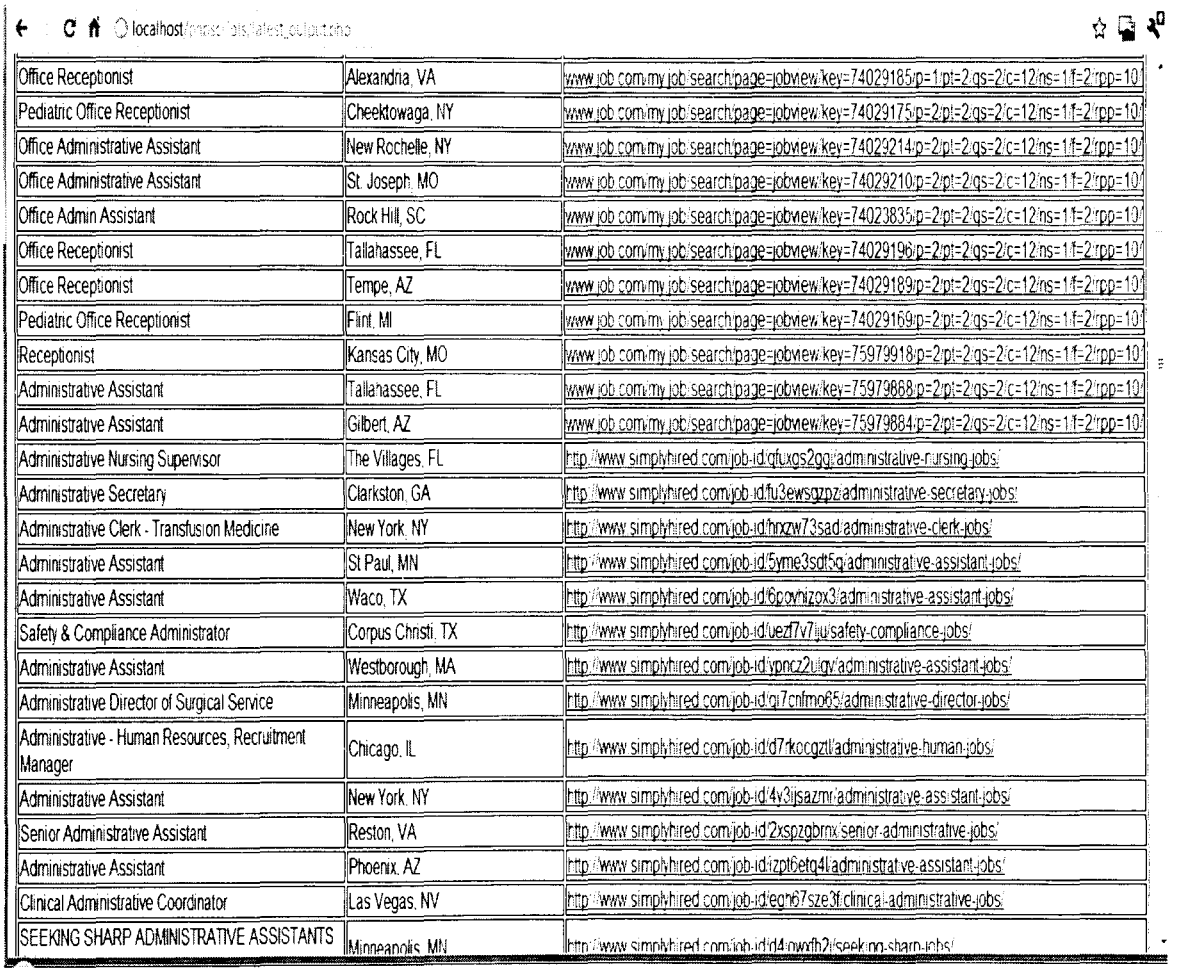


Figure 5.5. Feedback Message

Figure 5.6 displays a partial result from two different websites after initial wait time. Job from the website simplyhired.com [6] is appending to the initial set of administrative jobs.



Office Receptionist	Alexandria, VA	www.job.com/my.job/search/page=jobview/key=74029185/p=1/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Pediatric Office Receptionist	Cheektowaga, NY	www.job.com/my.job/search/page=jobview/key=74029175/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Office Administrative Assistant	New Rochelle, NY	www.job.com/my.job/search/page=jobview/key=74029214/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Office Administrative Assistant	St. Joseph, MO	www.job.com/my.job/search/page=jobview/key=74029210/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Office Admin Assistant	Rock Hill, SC	www.job.com/my.job/search/page=jobview/key=74023835/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Office Receptionist	Tallahassee, FL	www.job.com/my.job/search/page=jobview/key=74029196/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Office Receptionist	Tempe, AZ	www.job.com/my.job/search/page=jobview/key=74029189/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Pediatric Office Receptionist	Flint, MI	www.job.com/my.job/search/page=jobview/key=74029169/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Receptionist	Kansas City, MO	www.job.com/my.job/search/page=jobview/key=75979918/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Administrative Assistant	Tallahassee, FL	www.job.com/my.job/search/page=jobview/key=75979868/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Administrative Assistant	Gilbert, AZ	www.job.com/my.job/search/page=jobview/key=75979884/p=2/pt=2/qs=2/c=12/ns=1/f=2/rpp=10/
Administrative Nursing Supervisor	The Villages, FL	http://www.simplyhired.com/job-id/qfuxgs2gq/administrative-nursing-jobs/
Administrative Secretary	Clarkston, GA	http://www.simplyhired.com/job-id/fu3ewsqzpz/administrative-secretary-jobs/
Administrative Clerk - Transfusion Medicine	New York, NY	http://www.simplyhired.com/job-id/hrxzw73sad/administrative-clerk-jobs/
Administrative Assistant	St Paul, MN	http://www.simplyhired.com/job-id/5yme3sdt5q/administrative-assistant-jobs/
Administrative Assistant	Waco, TX	http://www.simplyhired.com/job-id/6povnizox3/administrative-assistant-jobs/
Safety & Compliance Administrator	Corpus Christi, TX	http://www.simplyhired.com/job-id/uezf7v7juu/safety-compliance-jobs/
Administrative Assistant	Westborough, MA	http://www.simplyhired.com/job-id/ypncz2ulqv/administrative-assistant-jobs/
Administrative Director of Surgical Service	Minneapolis, MN	http://www.simplyhired.com/job-id/qi7cnfmo65/administrative-director-jobs/
Administrative - Human Resources, Recruitment Manager	Chicago, IL	http://www.simplyhired.com/job-id/d7rkocgztl/administrative-human-jobs/
Administrative Assistant	New York, NY	http://www.simplyhired.com/job-id/4y3jsazmr/administrative-assistant-jobs/
Senior Administrative Assistant	Reston, VA	http://www.simplyhired.com/job-id/2xspzgbrnx/senior-administrative-jobs/
Administrative Assistant	Phoenix, AZ	http://www.simplyhired.com/job-id/zpt6etq4l/administrative-assistant-jobs/
Clinical Administrative Coordinator	Las Vegas, NV	http://www.simplyhired.com/job-id/egni67sze3f/clinical-administrative-jobs/
SEEKING SHARP ADMINISTRATIVE ASSISTANTS	Minneapolis, MN	http://www.simplyhired.com/job-id/rd4nwxfh2/seeking-sharp-jobs/

Figure 5.6. Retrieved Jobs from Job.com [7] and Simplyhired.com [6]

5.1. System Performance

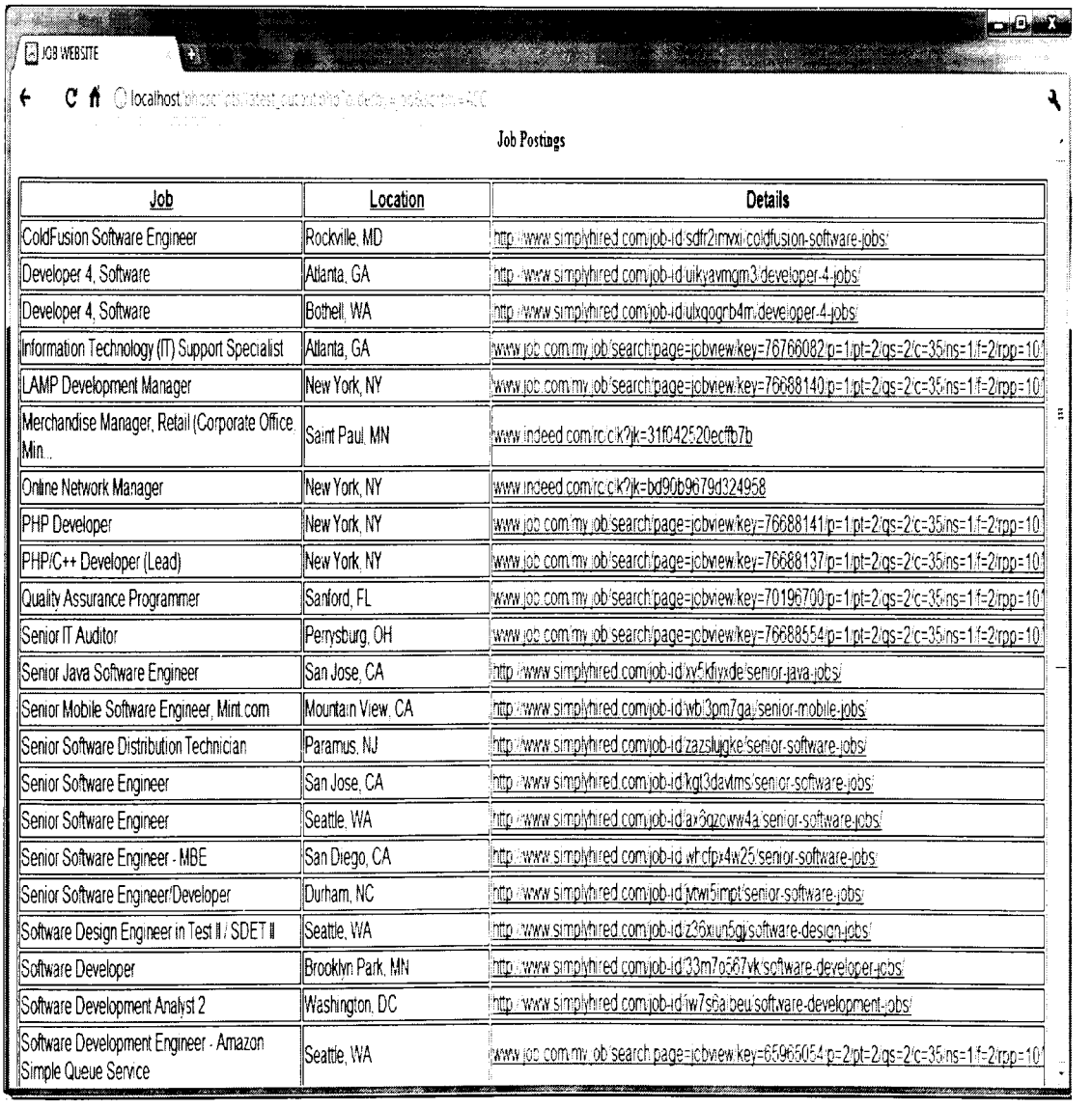
The performance of the system is tested based on the following criteria:

- Usability
- Reduced browsing time

The system is intended to be simple and user-friendly. Due to the minimalist nature of the User Interface, the user can select the job category from a drop-down menu. Since the

search engines unify the results into a unified table format, the user is able to browse through the jobs without much research on the user interface.

The results illustrated in Figure 5.7 shows the jobs for Computer sector in a sorted manner. Results from multiple job board engines are combined together and sorted by job title. Sorting by location is also done.



Job	Location	Details
ColdFusion Software Engineer	Rockville, MD	http://www.simplyhired.com/job-id/sdfr2rnwxi/coldfusion-software-jobs/
Developer 4, Software	Atlanta, GA	http://www.simplyhired.com/job-id/uikyavmgm3/developer-4-jobs/
Developer 4, Software	Bothell, WA	http://www.simplyhired.com/job-id/ulxqogrb4m/developer-4-jobs/
Information Technology (IT) Support Specialist	Atlanta, GA	www.jpc.com/my.job/search/page=jobview/key=76766082/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
LAMP Development Manager	New York, NY	www.jpc.com/my.job/search/page=jobview/key=76688140/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
Merchandise Manager, Retail (Corporate Office Min...	Saint Paul, MN	www.indeed.com/rc/clk?jk=31f042520ecffb7b
Online Network Manager	New York, NY	www.indeed.com/rc/clk?jk=bd90b9679d324958
PHP Developer	New York, NY	www.jpc.com/my.job/search/page=jobview/key=76688141/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
PHP/C++ Developer (Lead)	New York, NY	www.jpc.com/my.job/search/page=jobview/key=76688137/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
Quality Assurance Programmer	Sanford, FL	www.jpc.com/my.job/search/page=jobview/key=70196700/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
Senior IT Auditor	Perrysburg, OH	www.jpc.com/my.job/search/page=jobview/key=76688554/p=1/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/
Senior Java Software Engineer	San Jose, CA	http://www.simplyhired.com/job-id/xv5kflyxde/senior-java-jobs/
Senior Mobile Software Engineer, Mint.com	Mountain View, CA	http://www.simplyhired.com/job-id/wb13pm7ga/senior-mobile-jobs/
Senior Software Distribution Technician	Paramus, NJ	http://www.simplyhired.com/job-id/zazslugke/senior-software-jobs/
Senior Software Engineer	San Jose, CA	http://www.simplyhired.com/job-id/kgt3davtms/senior-software-jobs/
Senior Software Engineer	Seattle, WA	http://www.simplyhired.com/job-id/ax0qzcww4a/senior-software-jobs/
Senior Software Engineer - MBE	San Diego, CA	http://www.simplyhired.com/job-id/wfcfpx4w25/senior-software-jobs/
Senior Software Engineer/Developer	Durham, NC	http://www.simplyhired.com/job-id/ytwt5impt/senior-software-jobs/
Software Design Engineer in Test II / SDET II	Seattle, WA	http://www.simplyhired.com/job-id/z36xtun5gj/software-design-jobs/
Software Developer	Brooklyn Park, MN	http://www.simplyhired.com/job-id/33m7o367vk/software-developer-jobs/
Software Development Analyst 2	Washington, DC	http://www.simplyhired.com/job-id/w7s5a1beu/software-development-jobs/
Software Development Engineer - Amazon Simple Queue Service	Seattle, WA	www.jpc.com/my.job/search/page=jobview/key=65965054/p=2/pt=2/qs=2/c=35/ns=1/f=2/rpp=10/

Figure 5.7. Simplistic View of Job Results

The total time taken for operating JASE from the time of entering the website, selecting a job category and retrieving jobs is done. This is compared with the time taken to access and browse the three individual websites that are target websites to the search engine. Novice users were asked to perform this task. Table 5.1 shows the results of the total time taken for retrieving jobs.

Job Sector	**Time taken for Job Aggregation Search Engine(minutes)**	**Time taken for visiting individual target websites(minutes)**
Accounting	0.37	5.05
Administrative	0.30	3.22
Finance	0.31	1.36
Computer/IT	0.29	6.34
Average	**0.34**	**4.11**

Table 5.1. Total Time Taken for Job Retrieval from Target Websites

Results show that JASE is user friendly and takes much less time to search for an appropriate job rather than visiting numerous websites to find a job that interest us.

6. TESTING

Testing the result is an important aspect in the job retrieval program. Care should be taken that the correct text is being retrieved from the websites. Since data from different websites are being merged in a consistent manner, there are chances that data gets displayed at the wrong columns. Hence the crawler program and the parser program have been tested to check consistency. There are two level of testing.

1. The first stage of testing is done before and after the extracted data is inserted into the database from where it is retrieved by the front end program PHP and displayed to the users.
2. The second stage of testing is done after the job data gets displayed on the web browser. Comparison is made between the source job search engine webpage and the extracted data.

6.1. Testing for Crawler Program

The objective of testing the crawler program is to determine if the correct hot-links are identified and crawled from the target search engine webpage. This is done before the job data that is scraped from the websites is sent to the database. A log of the crawled websites is printed on the windows command prompt. The command to execute the Perl crawler script is passed with an argument which equals the job sector.

Test Case 1:

Input - Seed URL = *http://www.simplyhired.com/a/jobs/list/q-accounting/fdb-1*

Perl command = perl jobs2.pl *$admin_category*

where jobs2.pl is the crawler program for simplyhired.com , $admin_category = 1

Expected Output: Individual job listing hot-links shown in Figure 6.1 crawled. The page displays the main seed URL for the crawler program and the first page of the job listing.

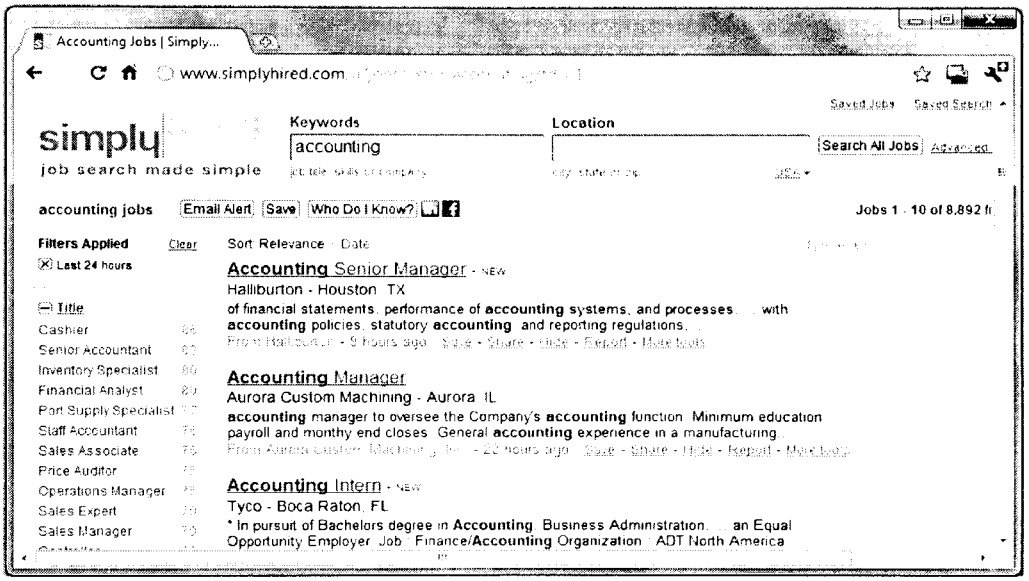


Figure 6.1. Screenshot of Simplyhired.com [6] Accounting Webpage

Actual Output: The crawled websites is printed on the windows command prompt.

Figure 6.2. Testing for Correctness of Crawled URLs

The list of all job titles and corresponding details page is printed. The total number of pages retrieved in the website is displayed. For prototype purpose, the total pages to be

scraped are limited to two. The log displays the page and the corresponding link that is currently being crawled.

Test Result - Comparing Figure 6.1 and Figure 6.2 shows that the correct URL has been crawled. The next page is crawled which is indicated by the print statement "Loaded page 2/2 of the jobboard" along with the URL which has been manipulated using the crawler.

Test Case 2:

Input:*http://www.job.com/my.job/search/page=results/pt=2/qs=2/c=12/ns=1/f=60/rpp=10/*

Perl command = perl jobs1.pl *$admin_category*

where jobs1.pl is the crawler program for job.com and $admin_category = 2

Expected Output: Figure 6.3 shows the expected output, i.e. the jobs that are listed for the administrative sector of job.com [7].

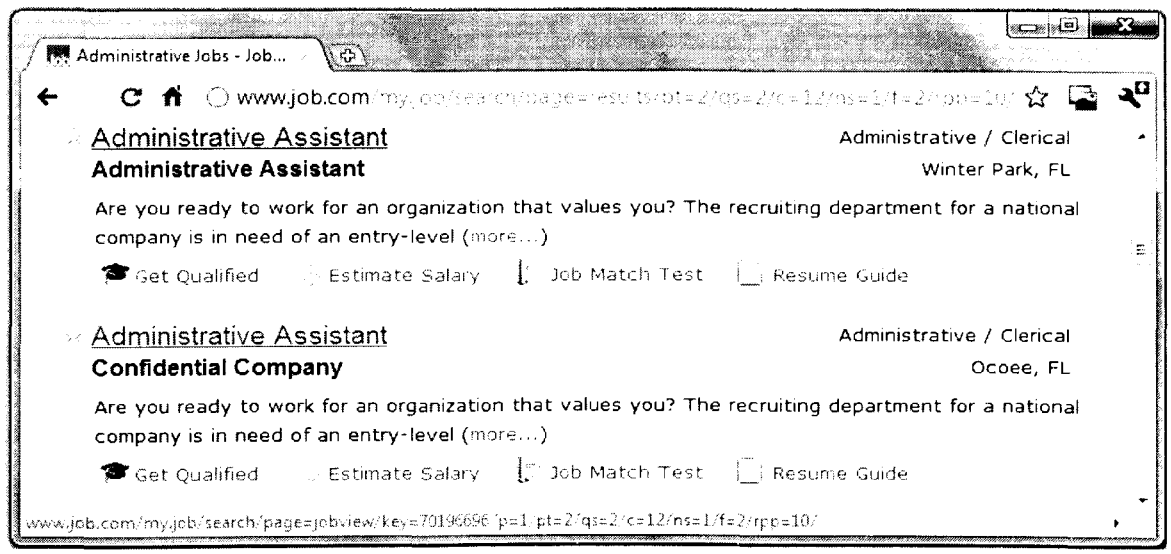


Figure 6.3. Screenshot for Administrative Sector of Job.com [7]

Actual Output: The actual result i.e., the URLs that are crawled for the administrative sector of job.com is show in Figure 6.4.

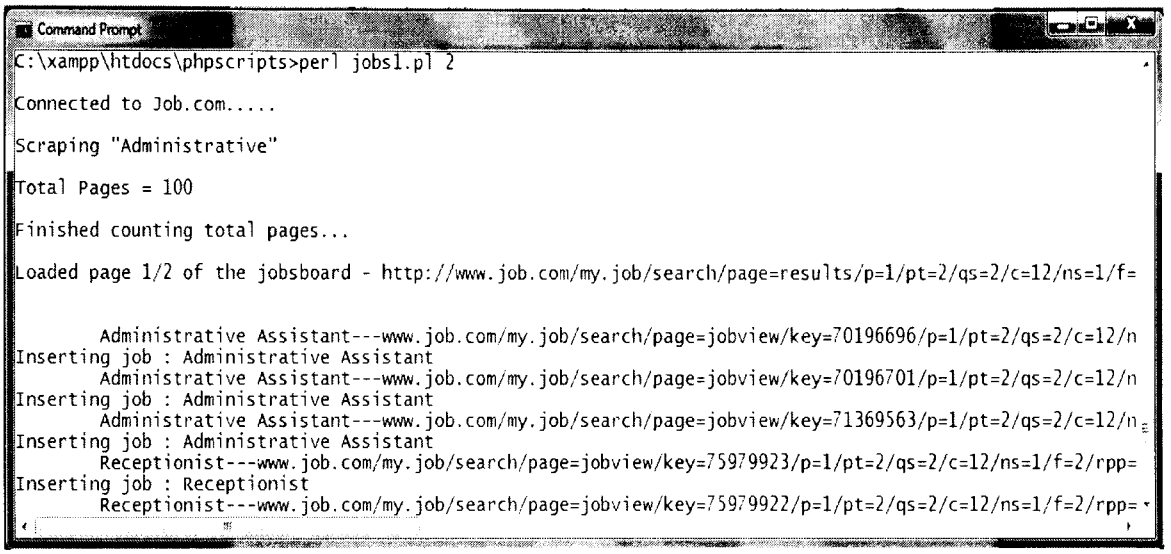


Figure 6.4. Crawled URLs for Administrative Sector of Job.com [7]

Test Result: On comparing Figure 6.3 and Figure 6.4, there is a match between the main URL of job.com [7] and the webpages identified by the crawler program. Hence the output is achieved.

6.2. Testing for Parser Program

The objective of testing the parser program is to test for consistency before data is inserted into the database. The parser program extracts job data from he webpages that are downloaded by the crawler and prints it on a notepad.

Test Case 1:

Input = Job listing page from Indeed.com as shown in Figure 6.5.

Expected Output = Job Title, Location and hotlink for each job extracted onto a notepad file.

A sample job listing page from Indeed.com is shown in Figure 6.5.

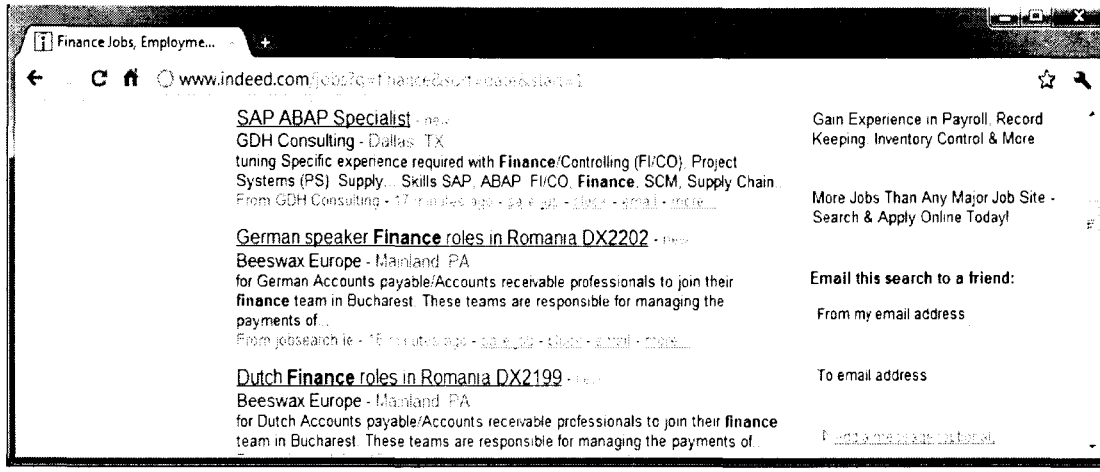


Figure 6.5. Screenshot for Indeed.com [5] Finance Sector

Actual output: Job data that is printed to a notepad file as shown in Figure 6.6.

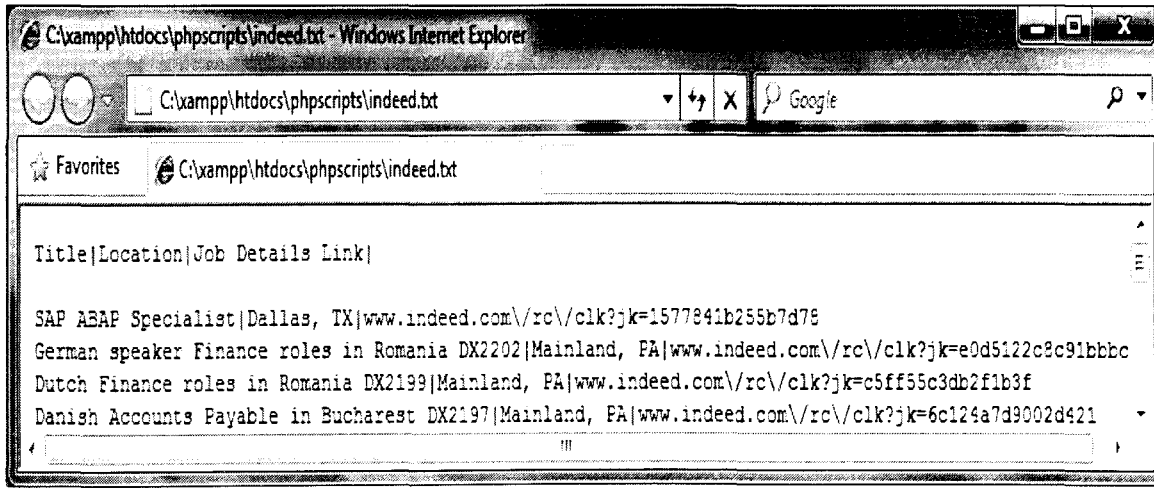


Figure 6.6. Screenshot for Indeed.com [5] Job Details Parsed to Notepad

Test Result: Printing the output in a notepad file is an important testing step. It shows that the data extracted is consistent with the source website. If there is an inconsistency, it shows that the parser is capturing incorrect HTML tags. Hence, further analysis is done to include the correct HTML tags which enclose the job data.

6.3. Functional Testing

Functional testing is done based on an integrated system level taking into account the functional aspects of the job retrieval engine. Given an input based on the specifications, the performance of the system is tested as an output on the graphical user interface. An input of job sector is given and the results that are expected are the segregated job details for the selected sector.

Test Case 1

The purpose of the test is to check the functionality of job sector drop-down menu.

Input: Selecting the drop-down menu list from the user interface.

Expected output: List of four job sectors namely Accounting, Administrative, Finance and Computer/IT.

Actual Output: Screenshot of job sector as shown in Figure 6.7 which displays the job disciplines.

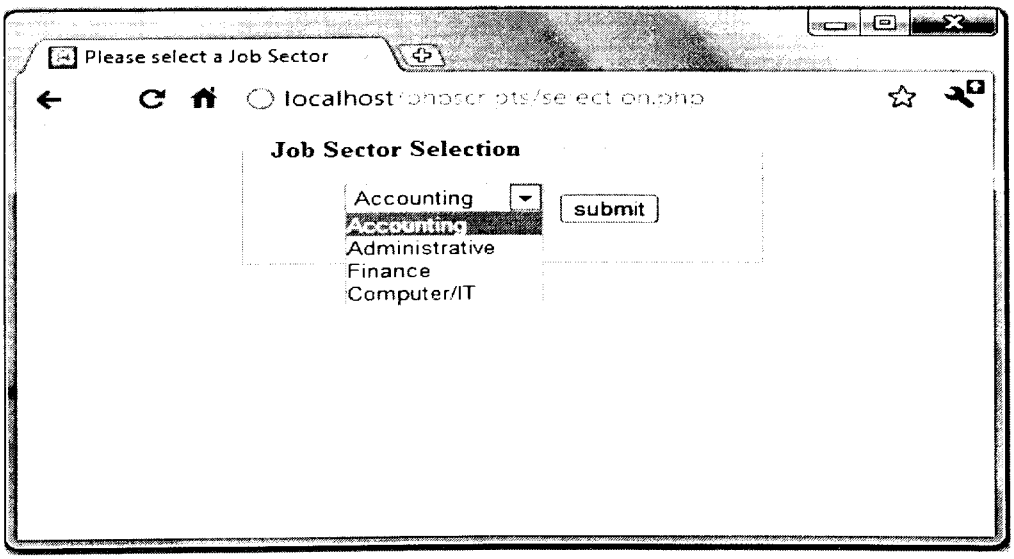


Figure 6.7. Screenshot of the User Interface – Drop-Down Menu

Test Case 2

Once the user selects the job sector, jobs are retrieved from three target search engines. The user is given a feedback to wait for more jobs while the crawler initially lists the jobs from one target source and the other target sources are being crawled.

Input: User submit action after selecting a job sector.

Expected Output: Feedback displayed at top of the job listing webpage.

The jobs retrieved from job.com [7] for the Computer/IT sector are initially displayed on the screen so as to minimize the users wait time. The remaining list of jobs is displayed on a periodic basis as the crawler crawls through the other job search engines.

Actual Output: Feedback for the Computer/IT sector is shown in Figure 6.8.

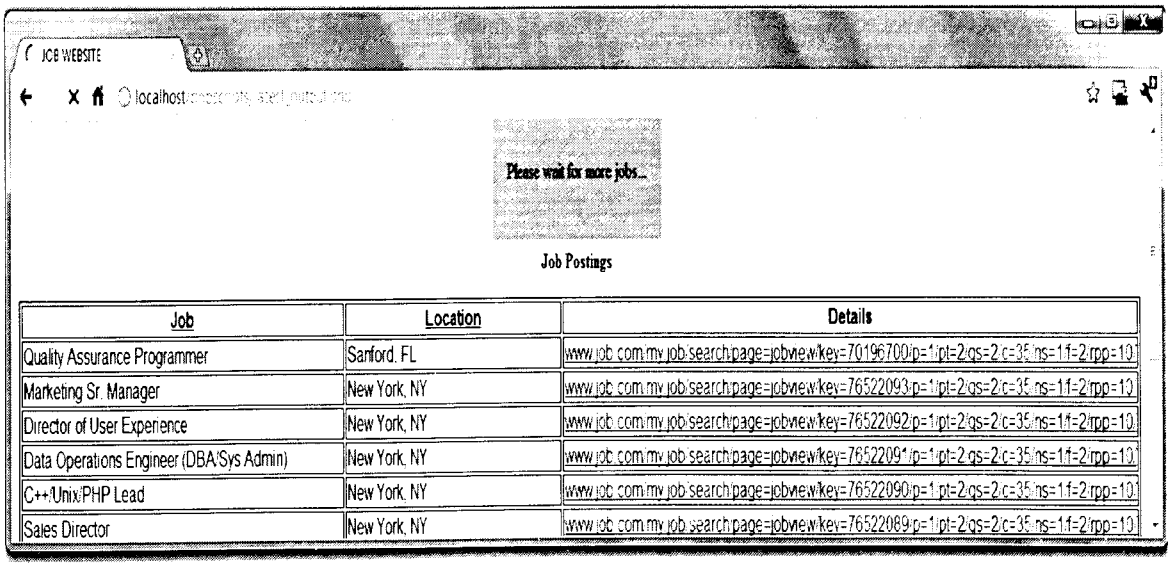


Figure 6.8. Screenshot of Initial Listing of Jobs for Computer/IT Sector

Test case 3

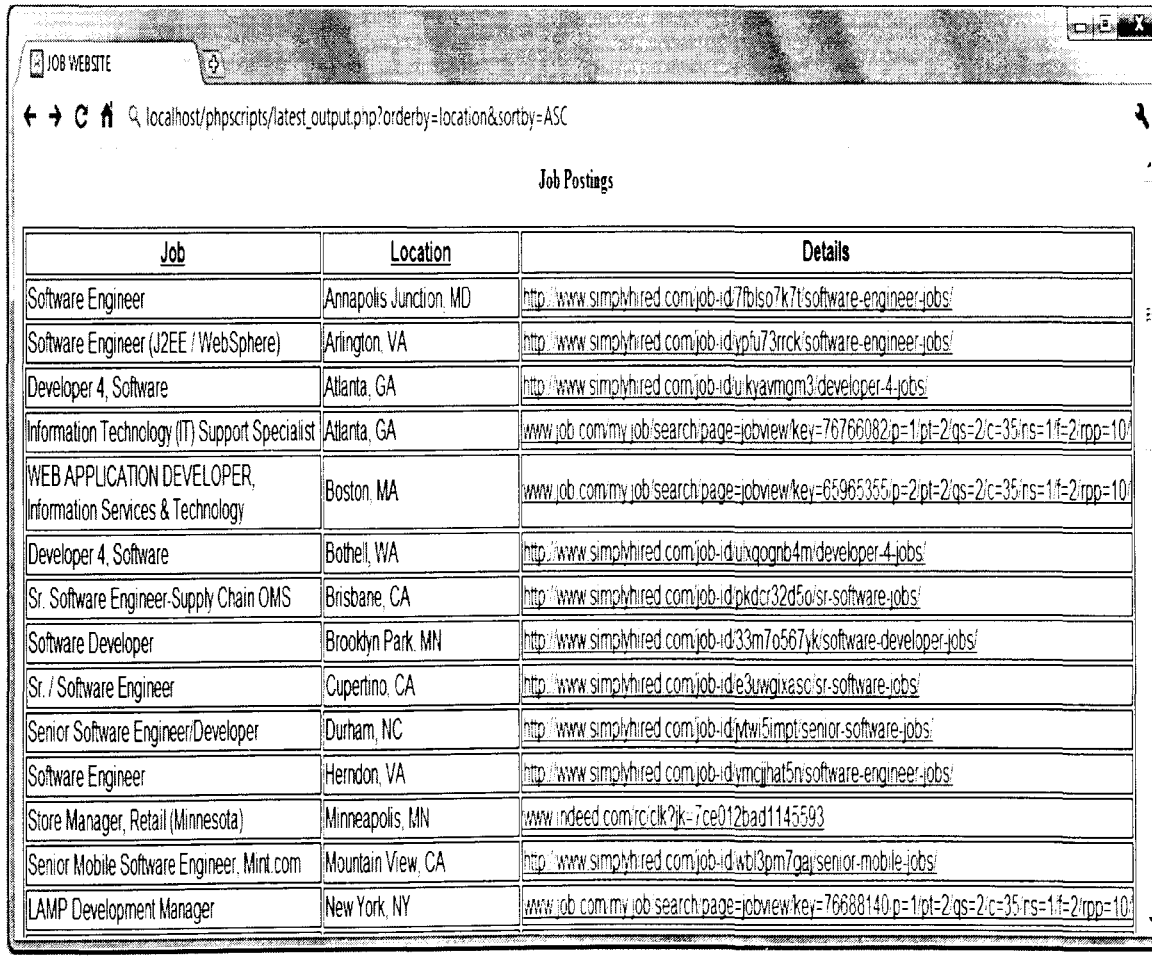
The objective of the test is to check the functionality of sort by location.

Input: Sector = Administrative and Sorting criteria = Location

Expected output: Location of jobs sorted in ascending order.

The administrative sector is chosen as the job sector. Once all the jobs are retrieved from the target job board search engines, the location is sorted in no particular order. On hitting the Location column heading, a sort is done in descending. Another hit on Location heading sorts the list in ascending order.

Actual Output: The display page containing the unified results for Administrative sector is illustrated in Figure 6.9.



Job	Location	Details
Software Engineer	Annapolis Junction, MD	http://www.simplyhired.com/job-id/7fblso7k7t/software-engineer-jobs/
Software Engineer (J2EE / WebSphere)	Arlington, VA	http://www.simplyhired.com/job-id/yp7u73rrck/software-engineer-jobs/
Developer 4, Software	Atlanta, GA	http://www.simplyhired.com/job-id/ukyavmqm3/developer-4-jobs/
Information Technology (IT) Support Specialist	Atlanta, GA	www.job.com/my.job/search;page=jobview/key=76766082/p=1/pt=2/qs=2/c=35/rs=1/f=2/rpp=10/
WEB APPLICATION DEVELOPER, Information Services & Technology	Boston, MA	www.job.com/my.job/search;page=jobview/key=65965355/p=2/pt=2/qs=2/c=35/rs=1/f=2/rpp=10/
Developer 4, Software	Bothell, WA	http://www.simplyhired.com/job-id/uxqognb4m/developer-4-jobs/
Sr. Software Engineer-Supply Chain OMS	Brisbane, CA	http://www.simplyhired.com/job-id/pkdcr32d5o/sr-software-jobs/
Software Developer	Brooklyn Park, MN	http://www.simplyhired.com/job-id/33m7o567yk/software-developer-jobs/
Sr. / Software Engineer	Cupertino, CA	http://www.simplyhired.com/job-id/e3uwgtxasc/sr-software-jobs/
Senior Software Engineer/Developer	Durham, NC	http://www.simplyhired.com/job-id/ptwi5impt/senior-software-jobs/
Software Engineer	Herndon, VA	http://www.simplyhired.com/job-id/ymcjhat5n/software-engineer-jobs/
Store Manager, Retail (Minnesota)	Minneapolis, MN	www.indeed.com/rc/clk?jk=7ce012bad1145593
Senior Mobile Software Engineer, Mint.com	Mountain View, CA	http://www.simplyhired.com/job-id/wbl3pm7ga/senior-mobile-jobs/
LAMP Development Manager	New York, NY	www.job.com/my.job/search;page=jobview/key=76688140/p=1/pt=2/qs=2/c=35/rs=1/f=2/rpp=10/

Figure 6.9. Screenshot of the User Interface with Jobs Sorted by Location

7. CONCLUSION

This paper has introduced a technique to connect jobseekers with real time jobs posted across top job boards. With many Meta search engines available and many job boards for job seekers to go to, it often becomes time consuming for the user to be the first to apply for a newly posted job on a given day at a given time. It often also becomes overwhelming for the user to visit each website if he is not able to find a particular job on a website.

This paper categorizes jobs by discipline, aggregates jobs from top job boards and brings in very recently posted jobs at the time when the user clicks on a particular discipline thus providing a platform for users to have access to new jobs posted across multiple job boards. Users are given information about the location of the job and a link to the detailed description if the job interests them.

The overall browsing time of the user is reduced on searching for the jobs of interest using JASE rather than visiting multiple websites for the same job criteria. Also, the overwhelming factor associated with browsing through various job websites is reduced since a unified list of jobs is displayed on the simple and user-friendly interface.

8. FUTURE WORK

The paper deals with aggregating jobs from multiple job search engines. Due to the fact that these search engines crawl from the web, there may occur the problem of job duplication. Hence, the paper can be extended to eliminate the problem of duplication.

Since the job retrieval program crawls over multiple websites in real-time, there is an increase in total time for retrieving the jobs. Hence, the program can be optimized for decreasing the total crawl time.

The number of job board websites crawled by the job retrieval program is limited. It can be increased to a larger number to give a larger number of results.

REFERENCES

1. Weiyi Meng, Clement Yu, and King-Lup Liu. 2002, "Building efficient and effective metasearch engines," *ACM Comput. Surv.* pp.48-89, March 2002
2. Eric J. Glover, Steve Lawrence, William P. Birmingham, and C. Lee Giles, "Architecture of a metasearch engine that supports user information needs" In *Proceedings of the eighth international conference on Information and knowledge management* (CIKM '99), Susan Gauch (Ed.). ACM, New York, NY, USA, pp. 210-216, 1999
3. G. Almpanidis, C. Kotropoulos, and I. Pitas., "Combining text and link analysis for focused crawling - An application for vertical search engines", *Information Systems, 32*(6), pp. 886-908, September 2007
4. Dorn, J., Naz, T., "Integration of Job Portals by Meta-search", Enterprise Interoperability II, Part IV, Springer London, pp 401-412, 2007
5. Indeed.com
   www.indeed.com, Retrieved on 02-14-2011
6. Simplyhired.com
   www.simplyhired.com, Retrieved on 02-14-2011
7. www.job.com, Retrieved on 02-14-2011
8. XAMPP
   http://www.apachefriends.org/en/xampp-windows.HTML, Retrieved on 01-24-2011.
9. Castillo, Carlos, "Effective Web Crawling", ACM SIGIR Forum 55 Vol.39 No. 1, pp. 55-56. June 2005

10. Mechanize

http://search.cpan.org/dist/WWW-Mechanize/, Retrieved on 01-13-2011

11. TokeParser

http://search.cpan.org/~gaas/HTML-Parser-3.68/lib/HTML/TokeParser.pm, Retrieved on 01-16-2011

12. Criston Souza, Eduardo Laber, Caio Valentim, and Eduardo Cardoso, "A Polite Policy for Revisiting Web Pages". In *Proceedings of the 2007 Latin American Web Conference* (LA-WEB '07). IEEE Computer Society, Washington, DC, USA, pp. 128-135, 2007

APPENDIX A. SOURCE CODE

The source code for job extraction consists of the crawler program and the parser program. The source code for job.com [7] and simplyhired.com [6] is given below.

```
#!"C:\xampp\perl\bin\perl.exe"

#=============================================================================
#Code for job.com
#=============================================================================

# standard modules
require 'functions.pl';
use strict;
use HTML::Entities;
use WWW::Mechanize;
use HTML::TokeParser;
use DBIx::DWIW;

#Code for data extraction from www.job.com

my $array = '';
my $datafile = '';
my %category;

my $admin_category = $ARGV[0];


# CONNECTING TO DATABASE

my $db_user       = 'root';
my $db_pass       = 'secret';
my $db_name       = 'job';
my $server        = 'localhost';
my $conn          = '';


=============================================================================
#ESTABLISH DATABASE CONNECTION#

$conn = DBIx::DWIW->Connect(User => $db_user, Host => $server, Pass => $db_pass,
DB => $db_name) or die "$!";
```

```
my $insertHandle = $conn->Prepare ("INSERT INTO jobs
                                                    (
                                                        sector,
                                                        website,
                                                        job,
                                                        company,
                                                        location,
                                                        Details
                                                    )

                                                        VALUES

                                                        (?,'Job',?,?,?,?)

                                                    ") or die "$!";



#============================================================
#Crawler Code
#============================================================

        $category{'1'}{'name'} = 'Accounting';
        $category{'1'}{'category_id'} = '4';          #sector id to be used in url of the website

        $category{'2'}{'name'} = 'Administrative';
        $category{'2'}{'category_id'} = '12';

        $category{'3'}{'name'} = 'finance';
        $category{'3'}{'category_id'} = '14';

        $category{'4'}{'name'} = 'computer';
        $category{'4'}{'category_id'} = '35';


        my $website1 = 'www.job.com';
        my $nextpage = 'http://www.job.com/';

# Initialise a new Mechanize instance for the main page function
        my $page = WWW::Mechanize->new();
        $page->agent_alias('Windows Mozilla');

        print "\nConnected to Job.com..... \n";
```

```
print "\nScraping \"".$category{$admin_category}{'name'}."\"\n\n";

my %jobs;
my $job;

my $location;

my $job_counter= 1;
my $counter = 0;
my $counter1 = 1;
my $numpages = 0;
my $total_pages = 0;    #it is not the total number of page but the existing page.
my $desc_page;
my $datafile = '';


$datafile = "job.txt";
open OUT, ">$datafile";
print OUT "Job Title|Location|Link\n";

# ==================== Initial Scraping==============================

$nextpage='http://www.job.com/my.job/search/page=results/p='.$counter1.'/pt=2/qs=2/c='.$
category{$admin_category}{'category_id'}.'/ns=1/f=2/rpp=10/';


        $page->get($nextpage);
```

admin_category: This section correctly identifies the number of pages required to get all links on the jobsboard.

```
        my $page_content= $page->content;

#=================================================================
#Parser Code
#=================================================================

    # Initialise a new TokeParser instance
      my $stream = new HTML::TokeParser(\$page_content);
```

```
while(my $tag = $stream->get_tag('span'))
    {
        my ($temp, $first, $second) = '';

        if($tag->[1]{id} eq "dRes")

        {

$total_pages = $stream->get_trimmed_text('/span');      # Returns "Returns Jobs 1 - 10 of
188,429"
                ($temp,$total_pages) = split('than ', $total_pages);
                $total_pages =~ s/jobs//g;

                ($first, $second) = split(',', $total_pages);

                $total_pages = $first.$second;

                if($total_pages%10 == 0)
                {
                   $total_pages = $total_pages/10;
                   print "Total Pages = $total_pages\n";

                }
                else
                {
                   $total_pages = $total_pages/10;
                   $total_pages = int $total_pages;


                   $total_pages = $total_pages + 1;
                   print "Total Pages = $total_pages\n";


                }
                print "\n";
                last;

        }

    }


#$total_pages = 2;
```

```
print "Finished counting total pages...\n";

my $l = 0;
for($l=1;$l<=$total_pages;$l++)
{

$nextpage=
'http://www.job.com/my.job/search/page=results/p='.$l.'/pt=2/qs=2/c='.$category{$admin_c
ategory}{'category_id'}.'/ns=1/f=2/rpp=10/';

        $page->get($nextpage);

print "\nLoaded page ".$l."/".$total_pages." of the jobsboard - ".$nextpage." \n\n";

###NOTE: This while loop will extract all the links on the job summary page.

        my $page_content=  $page->content;

# Initialise a new TokeParser instance

        my $stream = new HTML::TokeParser(\$page_content);

$desc_page = '';
while(my $tag = $stream->get_tag('h2'))
        {

        if($tag->[1]{class} eq "jobTitle_results")

         {


#NOTE: scraping links here:
                $tag = $stream->get_tag('a');


                $desc_page    =    $tag->[1]{href};    #    /job/337095/head-of-it-
services?vsrc=1


#NOTE: scraping position of the job:
                        $jobs{$job}{'name'} = $stream->get_trimmed_text('/a');
```

```
                        $tag = $stream->get_tag('div');
                        $jobs{$job}{'location'}=$stream->  get_trimmed_text('/div');

                        if($desc_page =~ /my\.job/)
                        {
                                $jobs{$job}{'link'} = $website1.$desc_page;
                                $jobcounter++;
                                print "\n";
                                print "\t";
                                print $jobs{$job}{'name'}." ".$jobs{$job}{'link'};
                                print "\n";

print OUT
escape($jobs{$job}{'name'})."|".escape($jobs{$job}{'location'})."|".escape($jobs{$job}{'li
nk'})."\n";
                        }
                        #$counter1++;


                      if($desc_page =~ /my\.job/)
                      {
            foreach my $job (sort {$jobs{$a} cmp $jobs{$b} } keys %jobs)
                                {
                                   my $dbresult = $insertHandle->Execute(

                                   $admin_category,

                                   $jobs{$job}{'name'},
                                   '',

                                   $jobs{$job}{'location'},
                                   $jobs{$job}{'link'}

                                ) or die "$!";

            print 'Inserting job : '.$jobs{$job}{'name'};


                    }
                                %jobs = ();
                        }

                }

            }
```

```
        # Wait beteween 3 and 5 seconds before getting the next page.
          sleep int(rand(5 - 3)) + 3;

        }

my $dbresult = $insertHandle->finish();
```

---

2. CODE FOR SIMPLYHIRED.COM

```
==========================================================================

my $datafile = '';
my %category;

$category{'1'}{'name'} = 'accounting';

$category{'2'}{'name'} = 'administrative';

$category{'3'}{'name'} = 'software';


#==========================================================================

#JOB EXTRACTION FOR SIMPLYHIRED.COM

my $jobsboard_url = 'http://www.simplyhired.com/';
my $admin_category = $ARGV[0];

print 'User input is : '.$admin_category.   '**************************';

$datafile = "simplyhired.txt";
open OUT, ">$datafile";
print OUT "Job Title|Location|Detail Link\n\n";


my $insertHandle = $conn->Prepare ("INSERT INTO jobs
                                    (sector, website ,job, company, location, details)
                                    VALUES
                                    (?,'simplyhired',?,?,?,?)
                                    ") or die "$!";
my $site2 = 'http://www.simplyhired.com/';

my $nextpage = 'http://www.simplyhired.com/';
```

```
# Initialise a new Mechanize instance for the main page function, and the subpage function
my $page = WWW::Mechanize->new();
$page->agent_alias('Windows Mozilla');

print "\nConnected to Simply Hired..... \n";

print "\nScraping \"".$category{$admin_category}{'name'}."\"\n\n";

my %jobs;
my $job;

my $job_counter= 1;
my $counter = 1;
my $counter1 = 1;
my $numpages = 0;
my $total_pages = 0;     #it is not the total number of page but the existing page.
my $desc_page;

my @jobtitles = ();
my @detaillinks = ();


# ================== Initial Scraping ============================

$nextpage =
'http://www.simplyhired.com/a/jobs/list/q'.$category{$admin_category}{'name'}.'/fdb-1/pn-
1';

print $nextpage."\n";

        $page->get($nextpage);

###NOTE: This section correctly identifies the number of pages required to get all links on
the jobsboard.

        my $page_content=  $page->content;

# Initialise a new TokeParser instance
        my $stream = new HTML::TokeParser(\$page_content);
#Loop to extract the total number of pages for the sector

            while(my $tag = $stream->get_tag('p'))
                {
                        my ($temp, $first, $second) = '';
```

```
                    if($tag->[1]{class} eq "job_counter")

                    {

                        $total_pages = $stream->get_trimmed_text('span');
                         ($temp,$total_pages) = split('of ', $total_pages);
                         ($first, $second) = split(',', $total_pages);
                         $total_pages = $first.$second;


                            if($total_pages%10 == 0)
                            {
                               $total_pages = $total_pages/10;
                               print "Total Pages = $total_pages\n";

                            }
                            else
                            {
                               $total_pages = $total_pages/10;
                              $total_pages = int $total_pages;
                             $total_pages = $total_pages + 1;
                             print "Total Pages = $total_pages\n";
                            }
                            print "\n";
                            last;

                    }

                }

$total_pages = 2;

print "Finished counting total pages...\n";

my $l = 1;
for($l=1;$l<=$total_pages;$l++)
{

        $nextpage = 'http://www.simplyhired.com/a/jobs/list/q-
'.$category{$admin_category}{'name'}.'/fdb-1'.'/pn-'.$l;


        $page->get($nextpage);
```

```
        print "\nLoaded page ".$l."/".$total_pages." of the jobsboard - ".$nextpage." \n\n";

###NOTE: This while loop will extract all the links on the job summary page.

        my $page_content=  $page->content;

# Initialise a new TokeParser instance
        my $stream = new HTML::TokeParser(\$page_content);

        while(my $tag = $stream->get_tag('div'))
        {

                if($tag->[1]{class} eq "more box")

                {


#NOTE : scraping links here :
                        $tag = $stream->get_tag('a');
                        $tag = $stream->get_tag('a');

                            if($tag->[1]{class} eq "permalink")

                            {
                            $desc_page = $tag->[1]{href}; # /job/337095/head-of-it-
services?vsrc=1
                            }


#NOTE: scraping position of the job:
                             $jobs{$job}{'name'} = $stream->get_trimmed_text('/a');
                             $tag = $stream->get_tag('a');
                             $tag = $stream->get_tag('a');

                            $jobs{$job}{'link'} = $desc_page;
                            ($jobs{$job}{'name'},$jobs{$job}{'comploc'}) = split(' at
',$jobs{$job}{'name'});
                            ($jobs{$job}{'company'},$jobs{$job}{'location'}) = split(' in ',
$jobs{$job}{'comploc'});
                            $jobcounter++;

                            print "\n";
                            print "\t";
                            print $jobs{$job}{'name'}."---".$jobs{$job}{'link'};
```

```
                print "\n";
                print OUT
escape($jobs{$job}{'name'})."|".escape($jobs{$job}{'company'})."|".escape($jobs{$job}{'l
ocation'})."|".escape($jobs{$job}{'link'})."\n";



                #push(@jobtitles, $jobs{$job}{'name'});
                #push(@detaillinks, $jobs{$job}{'link'});

        }

          #Inserting name, company, location and details link into the database
          foreach my $job (sort {$jobs{$a} cmp $jobs{$b} } keys %jobs)
          {

                my $dbresult = $insertHandle->Execute(
                                                $admin_category,
                                                $jobs{$job}{'name'}
                                        $jobs{$job}{'company'},
                                                $jobs{$job}{'location'},
                                                $jobs{$job}{'link'}
                                                ) or die "$!";

                #print 'Inserting job: '.$jobs{$job}{'location'};


          }
%jobs= ();

        }
}

my $dbresult = $insertHandle->finish();
```