Iowa State University
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2007

Internet data extraction based on automatic regular expression inference

Ye Lin
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation
Lin, Ye, "Internet data extraction based on automatic regular expression inference" (2007). *Retrospective Theses and Dissertations*. 14796.
https://lib.dr.iastate.edu/rtd/14796

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Internet data extraction based on automatic regular expression inference

by

Ye Lin

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Hui-Hsien Chou, Major Professor
Shashi K. Gadia
Xun Gu

Iowa State University

Ames, Iowa

2007

Copyright © Ye Lin, 2007. All rights reserved.

UMI Number: 1443109

UMI Microform 1443109

Copyright 2007 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

TABLE OF CONTENTS

LIST OF TABLES .... **iv**

LIST OF FIGURES .... **v**

ABSTRACT .... **vi**

CHAPTER 1. INTRODUCTION **1**

CHAPTER 2. BACKGROUND AND MOTIVATION **2**

2.1 Web Browser 2

2.2 Search Engine 3

2.3 Spider 4

2.4 Vect 5

CHAPTER 3. METHODS OF PROGRAM **8**

3.1 Search and View Web Pages 9

3.1.1 Use Perl to Load and Search Specific Web Pages 9

3.1.2 Design of the Spider 10

3.1.3 HTML Browser 14

3.2 Methods for Selection 15

3.2.1 Regular Expression 15

3.2.2 Methods for Single Selection 18

3.2.3 Methods for Table Selection 25

3.2.4 Data Structure to Hold Selections 28

3.2.5 Insertion of Selections 28

3.2.6 Methods for Canceling Selections 30

3.3. Generating the Output View 33

3.4 Generating Perl Code 34

3.4.1 Combine The Regular Expressions From Single Selections 34

3.4.2 Completing the Regular Expression for the Table Selection 35

CHAPTER 4. REAL-WORLD EXAMPLE **36**

4.1 Web Browsing and Spider Searching 36

4.2 Data Selection. 40

4.3 Output Data 41

4.4 Generate a Perl Code 42

CHAPTER 5. CONCLUSION AND FUTURE WORK **43**

REFERENCES **45**

ACKNOWLEDGEMENT **47**

LIST OF TABLES

Table 1: Structure tags used in Vnet. 17

LIST OF FIGURES

Figure 1: Snapshot of a Google web searching result.. 3

Figure 2: Snapshot of data selection in Vect Input Data panel.. 6

Figure 3: Snapshot of Vect Convert Data panel. 6

Figure 4: Structure of web pages visited by Vnet.. 10

Figure 5: Spider Structure in Vnet. 11

Figure 6: Snapshot for Vnet Search Data panel... 37

Figure 7: Snapshot for using spider search engine in Vnet. 39

Figure 8: Selections on Vnet Search Data panel.. 40

Figure 9: Output data view on Selected Date panel.. 41

Figure 10: Snapshot for running generated Perl program on Vnet... 42

ABSTRACT

Modern biologists face greater challenges for dealing with the enormous amount of biological data generated by a variety of modern biological studying methods. It becomes very important to extract and analyze specific information from these data efficiently. Nowaday, with the advantage of the World Wide Web, most of these data are available on the Internet. However, since these data are huge and diverse, to find and obtain these data on the Internet also is a daunting task. These tasks can be very difficult for researchers without computer programming skills to conduct, and they can also slow down the research and development process for many modern research areas.

In this thesis, we introduce an auto-programming tool Vnet (Visual Network data Extraction and program Tool) for biologists to help them obtain specific web pages on the Internet and extract useful information from them. This tool can observe the visual extraction of sample data by users via a graphical user interface, i.e., through the mouse point-and-click operations familiar to most computer users. After that, it will be able to generate computer programs that can automatically carry out the same kind of data download and extraction tasks for users. That is to say, by memorizing the ways a user data download and extraction data, this tool can turn that into computer solutions. Vnet does not require any sophisticated computer science training to use it, and it can do the programming job automatically.

CHAPTER 1. INTRODUCTION

Vnet is an auto-programming tool which can help users conduct the following jobs:

1) Search the Internet manually on Vnet browser or by the spider search engine provided by Vnet;

2) Select data from these web pages and compile the selections into regular expressions; and

3) Output selected data and generate Perl programs which embed the functions for the web searching and automatic information extraction.

The generated Perl programs can run independently. Users just need to specify several parameters as they gave to the Vnet spider search engine. The Perl programs will return the data which appear on the same areas as users have selected in Vnet browser.

This thesis is organized into the following chapters. Chapter 2 provides the background information on current methods and tools that can collect information on the Internet or extract useful information from a huge amount of data, which leads to the discussion of the motivations for creating Vnet. Chapter 3 introduces the methods used to implement the functions of Vnet which includes how to create the spider for Vnet, how to translate user selections to correct regular expression sequences, the types of selections Vnet handles and how Vnet generates output views and Perl programs. Chapter 4 presents how Vnet works by showing a demo. Finally, Chapter 5 summarizes the contributions of this work and suggests future expansion directions.

CHAPTER 2. BACKGROUND AND MOTIVATION

In many modern science and research fields, scientists such as biologists and biomedical researchers can reuse some experiment data generated previously by others and archived on some web pages for public access. By using or analyzing these data without repeating those experiments again, they may discover some important things or rules that are not noticed by the original researchers, or to follow up on prior work. The research advance process can be dramatically shortened by the interchange of data. However, the following issues prevent scientists from doing those tasks smoothly:

a) Since most often the data are stored with an enormous amount of similar data in an online database, it is difficult for researchers to find the right data to use;

b) The data could be huge and complicated for easy extraction and analysis; and

c) Most web sites update their database frequently, therefore the data that researchers have extracted previously may not be the latest version.

To avoid those conditions, scientists have to use some automatic tools to conduct these tasks and to update their data as needed. The following introduces some tools that can partially help them, but may be too complicated to achieve the purposes especially for non-programmers.

2.1 Web Browser

Since Internet has become more and more popular, viewing data from a web site within a web browser becomes an easy task for most of people in this decade [1]. Every kind of browsers, e.g., Microsoft Internet Explorer, Mozilla Firefox, Opera or Safari, can do this very simply by taking an URL or following some the hyperlinks on the WebPages, no matter what kind of operating system they are running on. However, until now, their major functions are still just to view the web pages; none of them have the capability to automatically traverse the

Internet all by themselves. The closest they can do is to link their searching field to some popular web search engines.

2.2 Search Engine

A search engine or search service helps find information stored on a computer system or on the World Wide Web. It can work inside a corporate or proprietary network or on a personal computer [2]. From the user's point of view, a search engine works under following process:

a) Users input some specific search criteria using some keywords, and ask for contents meeting them; and

b) The search engine then returns a list of references that match those criteria.

Most commonly, a search engine refers to a Web search engine, and it searches for information on the Internet. Google and Yahoo are two of the most popular web search engines. They both provide their web search services through their web pages. Users input and submit the search criteria to their web sites, and then they display the searching results on their web pages. Figure 1 is a screen snapshot of Google web searching result.

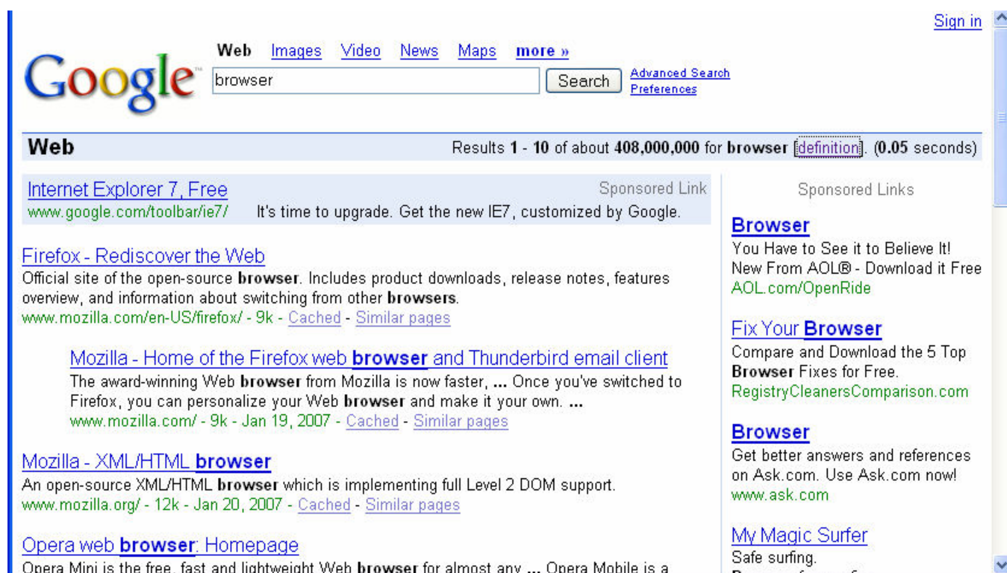

Figure 1: Snapshot of a Google web searching result.

Web search engines are very convenience. however, they still have some limitations:

1) Most of the time they return too much information, and most of the information are not what the users find useful. Sometimes, users need more specific searches on some specific known websites instead. Web search engine providers are working on this issue. Google provides a service called SiteSearch which enables users to search on some specific websites. Unfortunately, this service works only on static pages that can be directly linked from the Internet; it does not work on dynamically generated data.

2) For some websites that provide huge online databases like the Genbank, web search engines are not able to obtain all the information from these databases to make their index. Therefore, if users want to search more specific information from those websites, web search engines cannot be used and users must individually access those database websites.

3) Search engines cannot tell about the most current web pages. To provide fast searching for users they index billions of web pages routinely, but this indexing step takes a long cycle. Therefore, it is impossible for search engines to update all the indexed pages every minute. This is acceptable for most of the web pages that do not change after they are completed, but for some dynamic web pages that are updated very frequently, web search engines will not be able to give the most current results. People who want to keep track of the frequently updated web pages have to use some more complicated tools on their own. Spider is one of them.

2.3 Spider

A Web spider (also known as a Web crawler or Web robot) is an automated script or program which browses the website on the Internet in a predefined, automated process. This process is called Web crawling or spidering. Many legitimate websites, in particular search engines, use spidering as a method of providing up-to-date data. Spiders are mainly used to

create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. Spiders can also be used for automating Web site maintenance tasks, such as checking broken links or validating HTML code. In addition, spiders can be used to gather specific types of information from Web pages, such as collecting e-mail addresses (usually for spams).

The working process of a Web spider is as follows:

1) It starts with a list of URLs to visit, called the seeds;

2) As the spider visits these URLs, it identifies all the hyperlinks in the visited page and adds them to the list of URLs to visit; and

3) It recursively runs step 2 according to a set of policies and depth limits.

Spider is very powerful for web information collection. However, it needs some computer programming skill to implement, especially to set some special search requirements. Most of the Internet users do not have the ability to use this powerful tool.

2.4 Vect

Vect (the Visual Extraction and Conversion Tool) are developed by Dr. Hui-Hsien Chou [3]. Users can manipulate their sample data inside its data panels, and then Vect can generate Perl programs to repeat these tasks. Vect is able to satisfy the requirement to process textual format data. In this section, I will present how Vect works by showing a simple example.

Vect has four panels: Input data, Convert data, Output data and Perl Program. In this example we will extract and convert data from a Genbank report file. First, we load the data file to the Input data panel and select the required data. Vect provides many kinds of selection rules, e.g., single left click selects one column, right mouse drug defines a new line selection, etc.

Generally, users need to define a group of selections to mark the specific data they want selected. Figure 2 shows the screen snapshot after a group of selections was made:

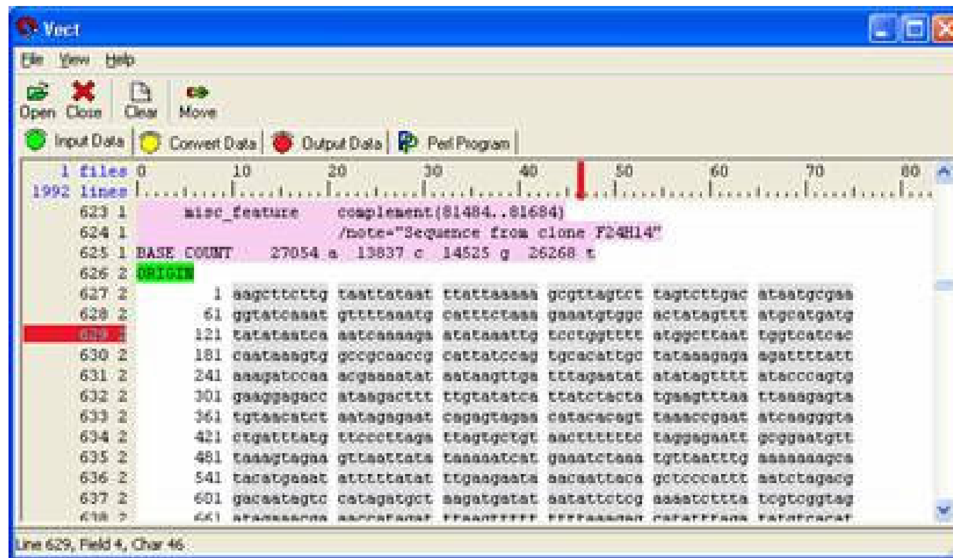

Figure 2: Snapshot of data selection in Vect Input Data panel.

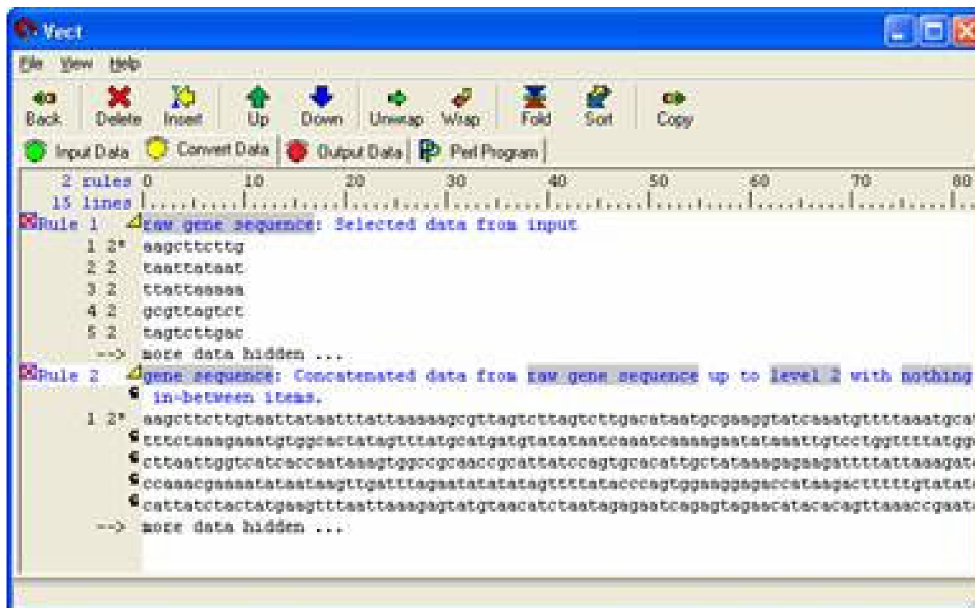

Figure 3: Snapshot of Vect Convert Data panel.

After the selection, in the second step users can click the move button to extract the selected data to the Convert data panel. In this panel, users can use many predefined rules to modify those data, e.g., the concatenate rule, quote rule, pattern rule, pick rule, merger rule, and so on.

Users can also define their own rules to modify those data. Users can see what the data look like after the conversion in Figure 3.

In the third step, users select a converted data set and click the move button again, so the data will be moved to the Output data panel. This panel gives users a chance to format and preview the final output data.

Finally, in the last step, users can compile the whole process into a Perl program for latter use in the Perl program panel. Since each Vect rule contains a part of Perl code in it, the combination of the rule set combines those Perl code with specific functionalities together. Therefore, Vect can generate a Perl program based on these code fragments. After the Perl program is generated, if users want to do the same kind of data extraction task on other files with similar format, they can just run the Perl program instead of having to do the same manual extraction process again.

Vect is very convenient and powerful on data extraction and conversion. However, it currently works on local data files only and has no Internet access capability. As mentioned earlier, most of time people obtained the data files they need to work with from some web sites such as the GenBank web site. This project was the result of the idea to combine some functions of Vect and spider together, and provide more direct data extraction convenience for users.

CHAPTER 3. METHODS OF PROGRAM

Vnet (Visual Network data Extraction and program Tool) is a software tool that has been created for searching and extracting specific information on some web pages. Its working process is as follows:

1) Users can either find web pages to visit manually similar to using a normal web browser, or use the Vnet spider engine to automatically visit some web pages:

1.1) If users want to find information manually, they can enter the web address into the URL text field and or click the left mouse on some hyperlinks appeared in some web pages. In this case Vnet works as a normal web browser.

1.2) If users want to use the spider engine, they need to input an URL, a few keywords and the search level. Vnet will return a sequence of satisfied web pages. Users can browse through them and find the web pages they really want to visit.

2) After users visit the web pages they expected, they can start to extract data from them. In other words, users can start to select data. There are two ways to select data. One is single right mouse click on some words, and the other is single right mouse click on some words in a table while holding the Shift key. The first method will highlight the block area that users selected to mark them as selected. The second method will highlight the column users selected and also mark the text in that column as selected. The information the position of each selection in the HTML source will be stored as a regular expression sequence for later use.

3) If users are not satisfied with some selections or selected some data by mistake, they can perform single right mouse click on a highlighted area and the selection of that area will be canceled. They can also clear all of the selections by clicking the "Clear" button.

4) When users completed the data selection, they can move the selected data to the next panel similar to Vect. In this panel, users can view the selected data and save them to a text file. If they are not satisfied with the result, they can clear the selections and go back to select from the HTML again.

5) After users finished all prior steps, they can create a Perl program in another panel. Vnet has the regular expression sequences that represent the position information of all user selected data that allow it to generate the Perl code. Users can run the Perl code in Vnet or independent of Vnet. In either case the Perl program will take the user specified URL, keywords and search level, conduct the search and extraction steps automatically and finally output the extracted data.

The following sections in this chapter present more detail information of the methods we used to implement the above working process.

3.1 Search and View Web Pages

3.1.1 Use Perl to Load and Search Specific Web Pages

To create a browser that can view the HTML web pages, we have to connect to web servers and download the HTML code from them. There are many options for this job in many programming langrages such as Java, C++, Perl, Visual Basic, and so on. All of them provide good libraries for network connection. However, we chose Perl in this project for the following reasons [4, 5, 6]:

a) Perl has the incredible ability to process text easily, with powerful features like regular expressions and hash data structures built-in.
b) Perl has many good modules (i.e., packages) for network access, HTML parsing, etc.
c) Perl is easy to learn and use.

This project focuses on the HTML code searching and extraction. Therefore, Perl is the best choice. Actually, since Perl is so easy to learn and use, many popular spider software tools are written in Perl, such as CoBWeb [7].

3.1.2 Design of the Spider

3.1.2.1 Input Parameters

There are three input parameters for the web spider in Vnet, which are the destination URL, keywords, and search level. Destination URL is the starting web address that users expected to find the information they want. Keywords are word strings which define the specific search criteria. Search level specifies how deep the users want to go in the webpage hierarchy. Figure 4 shows the structure of web pages Vnet might visit.

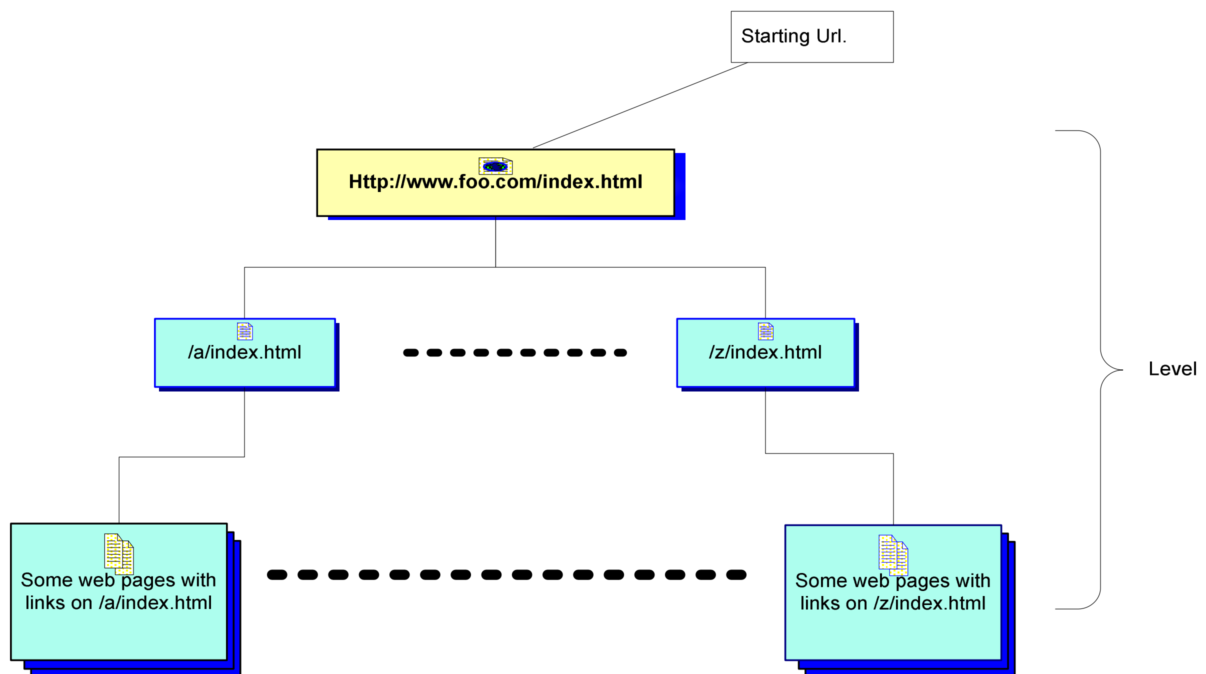

Figure 4: Structure of web pages visited by Vnet.

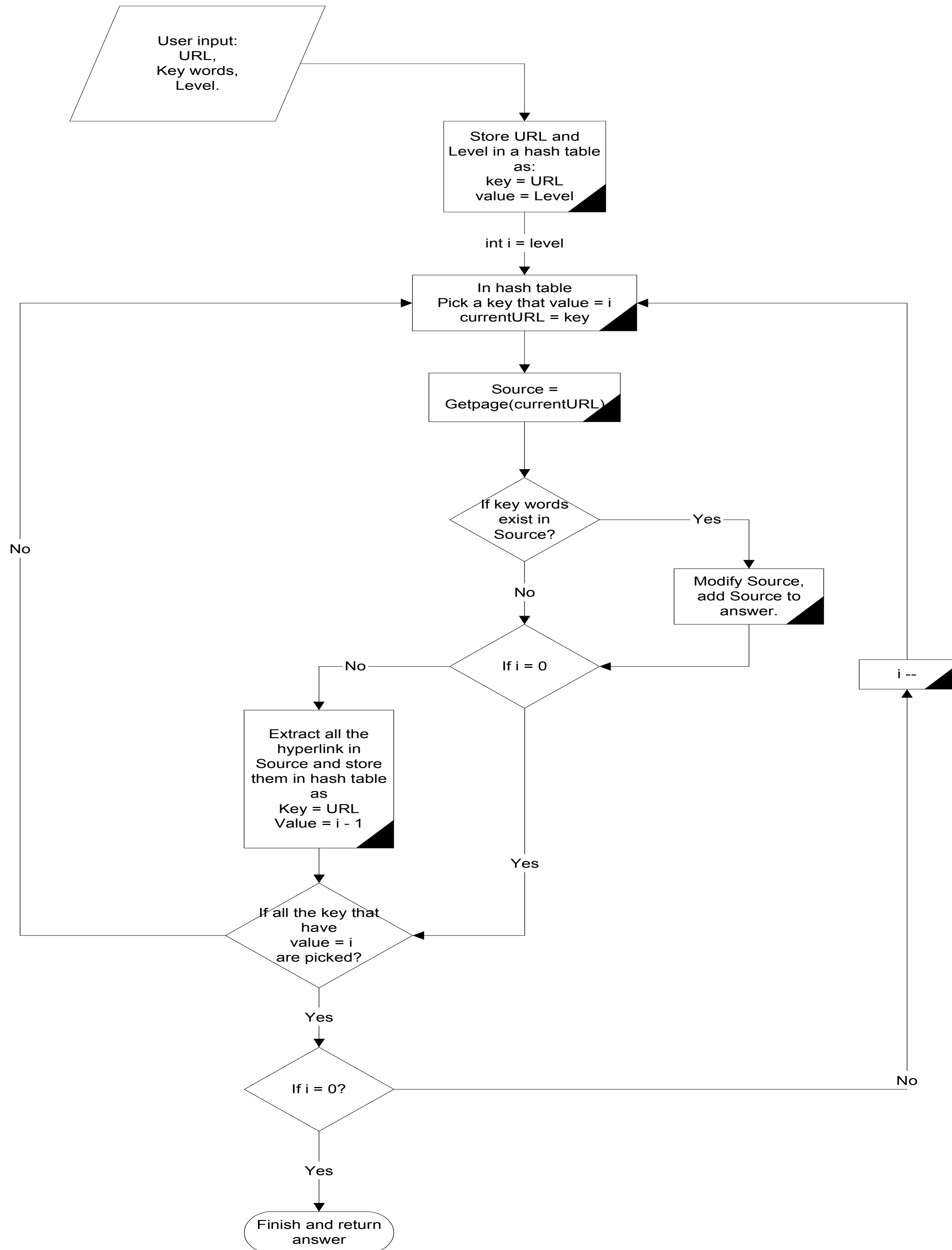

Figure 5: Spider Structure in Vnet.

3.1.2.2 Structure of The Spider Search Engine

Figure 5 shows the work process sequences of the spider search engine used in Vnet. There are two loops in this process. The outer loop is based on the level that user specified. It assures that Vnet searches to the bottom level web sites from user input URL. The inner loop is based on the size of the hash table. It makes sure that all the web sites in the specific level are visited, so basically this is a breath-first search.

Vnet first saves the user defined URL and level to the hash table, and then enters into the outer loop. At the beginning of the first run of the outer loop, there is only one URL in the hash table, but at the end of the first run, URLs with level = (i - 1) will be added to the hash table. For each cycle of the outer loop, Vnet will visit all the URLs with level = i in the hash table, and add more URLs with level = i – 1 in it till the last loop. Once the condition i == 0 is true, Vnet will not extract the links from the loaded web pages. For the inner loop, Vnet will visit each key in the hash table and load that URL if their level is i. The running time of the inner loop will increase with the size of the hash table.

In order to avoid visiting the same webpage more than once, before Vnet write an URL to the hash table it will do URL normalization and then check if the URL already exists in the hash table.

URL normalization process in Vnet includes the following jobs:

1) **Converting the scheme and the whole URL into lower case**. Since the scheme and host of URL are case insensitive, it is good to keep them both in lower case to reduce redundancy. However, for some operating system like UNIX, the path of an URL and the file name could be case sensitive. By the research of Sang, turning the URL into lower case reduce a big percentage of redundant web pages (52%), but only lost a very low percentage of unique

pages (1%) [8]. Therefore converting the whole URL string into lower case is worthy to do. Example:

`HTTP://Foo.org/AABB/foo.html → http://foo.org/aabb/foo.html`

2) **Removing the fragment**. Because an URL with and without the fragment represent the same resource, the fragment portion of a URL is removed. Example:

`http://foo.org/foo.html#section1 → http://foo.org/foo.html`

3) **Removing port 80**. Since the default http port is 80, it can be removed if specified in an URL. Example:

`http://foo.org:80/index.html → http://foo.org/index.html`

4) **Changing relative URL to absolute URL and removing ".." and "." segments.** The relative URLs will not point to the correct web address. Vnet convert them by using a Perl Package called `HTML::LinkExtor`. By giving a base URL, it will convert those URL automatically. Example:

`../abc.html → http://foo.org/abc.html`

`http://foo.org/../a/./b.html → http://foo.org/a/b.html`

5) **Remove URLs that are not rooted with the base URL** This requirement does not exist in the general URL normalization. However, for Vnet, since it is expected to visit only URLs under strict hierarchical relationship, we eliminate URLs that branch out of the hierarchy or to the other web servers.

Since our purpose is to extract text data, we filter pictures, java scripts and many other multimedia features from the HTML code. Users can then focus on the text data when using our program.

3.1.3 HTML Browser

There are many open source HTML browsers available such as Mozilla and Netscape. However, since our purpose is only to display and extract text from web pages, we need smaller and more efficient HTML browsers. wxHtmlWindow is one of them. wxHtmlWindow, designed by Vaclav Slavik, is a class within wxWidgets[9], a popular C++ GUI toolkit. It is designed to display the help or about information files within an application written using wxWidgets, using either the rich text format or HTML format. It handles most of the basic markup tags and structure tags. Furthermore, it provides pluggable tag handlers so that developers can create their own customized tag handlers.

Each word or text sequence that was shown on the wxHtmlWindow was contained by a container called wxHtmlCell. Each wxHtmlCell in the wxHtmlWindow has its coordinate and size values. When users click on some words on the wxHtmlWindow, the wxHtmlWindow will know which cell is clicked by comparing the coordinate values of the mouse clicking position. This property helps us to detect the data which the users intend to select.

3.2 Methods for Selection

3.2.1 Regular Expression

3.2.1.1 Definition

A regular expression is a meta string that describes or matches a set of normal strings, according to certain syntax rules [10]. Regular expressions are used by many text editors and utilities to search and manipulate bodies of text based on certain patterns. Many programming languages support regular expressions for string manipulation. For example, Perl and Tcl have a powerful regular expression engine built directly into their syntax. Here, we used Perl to achieve our purposes.

3.2.1.2 HTML (HyperText Markup Language)

HTML (HyperText Markup Language) is developed and maintained by W3C. It is not a forced standard like Latex or some other markup languages [11]. An incorrectly formatted HTML webpage can still be accepted and displayed normally by most of the popular web browsers. Using regular expressions to find some dynamic data on those poorly formatted web pages could be very difficult. However, since our purpose is to extract data from some database websites, and normally these websites are well formatted, our mission is still possible.

3.2.1.3 Why Regular Expression Can Query HTML Code.

HTML is a markup language. It uses tags to describe information such as structure, presentation, hypertext and so on for the text in the web pages. The following is a simple

example of the HTML code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 1.0 Transitional//EN">
<html>
    <head>
         <title>my example</title>
    </head>
    <body>
        <h1>This is a header</h1>
        <p>This is a paragraph. This is a paragraph. This is a paragraph.
        This is a paragraph. This is a paragraph. This is a paragraph.
        This is a paragraph. This is a paragraph. </p>
        <p>This is another paragraph. This is another paragraph.
        This is another paragraph. This is another paragraph.
        This is another paragraph. This is another paragraph.
        This is another paragraph. This is another paragraph.
        This is another paragraph. </p>
    </body>
</html>
```

In a browser, the above HTML code produces the following text:

This is a header

This is a paragraph. This is a paragraph. This is a paragraph. This is a paragraph. This is a paragraph. This is a paragraph. This is a paragraph. This is a paragraph.

This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph. This is another paragraph.

We found that the tags that represent the structure of the HTML code can help us locate specific text. People usually group related information in HTML blocks like paragraphs, tables and table cells. By identifying these special tags in an HTML code and creating a regular expression sequence based on them, we can obtain the information which are contained in some blocks. We explain more by showing the following example.

In the sample code, the <p> tags represent paragraphs. We can easily find the text in the first paragraph by searching the HTML code between the first <p> and </p>. In Perl, we can use a regular expression sequence ".*?<p>(.*?)</p>" to extract the text within these two tags

from the HTML source. For HTML code with more complex structures we may use more elaborated regular expression sequences to extract data. In HTML, many tags can define structure information. The tags in Table 1 are used in Vnet to detect the correct structure of HTML code and generate regular expression sequences [11]:

Tags Name	Description	Tags Name	Description
<blockquote>	Large quotation		List item
<body>	Document body		Ordered list
 	Line break	<p>	Paragraph
<dd>	Definition	<pre>	Preformatted text
<dir>	Directory list	<table>	Tables
<div>	Logical division	<tbody>	Table's body cell
<dl>	Definition list	<td>	Table cell
<form>	Input form	<tfoot>	Table's foot cell
<h1>	Header 1	<th>	Same as <thead>
<h2>	Header 2	<thead>	Table's header cell
<h3>	Header 3	<tr>	Table row
<hr>	Horizontal rule		Unordered list
<html>	HTML document		

Table 1: Structure tags used in Vnet.

For some tags like <h1>, they represent more than the structure information, but we treat them the same as structure tags.

3.2.1.3 Benefits of Using Regular Expressions.

When users use normal web browsers to browsing the Internet, they can also obtain the information they are interested by selecting, copying and pasting the data as what we do within Vnet, so what is the benefit of using regular expressions? Indeed, if all the selections were made only for one time use, then there is no reason to use regular expressions. However, there are cases that the same kinds of selections have to be made again and again against different web pages [12]. For example, people may have to visit same web sites several times a day to get the updated information. The information they are looking for generally will be posted in

the same locations on those websites. With the help of regular expressions, we can easily query the HTML code of those web pages, and extract the information that are shown on those special positions within those web pages.

There is another example. For some web sites, such as GenBank, lots of their web pages are dynamically generated by scripts. The data on those types of web pages are obtained from their database, therefore their web pages generally are created under the same format because they are generated by the same scripts. As a result, if we need to obtain similar information from these kinds of websites, we can use the same regular expressions to query them.

3.2.2 Methods for Single Selection

The process of single selection includes the following steps:

1) Find which wxHtmlCell is selected.

2) Obtain the position of the selection in the HTML code.

3) Obtain a sequence of tags that can represent the position.

4) Generate a regular expression sequence from the sequence of tags.

5) Obtain the group text from the HTML source using the regular expression sequence.

6) Mark the group of text that is selected by the regular expression sequence on the Vnet browser so users know what they have selected.

The following sections describe the details of these steps.

3.2.2.1 Find The wxHtmlCell That Is Clicked

As we described in section 3.1.3, each wxHtmlCell in the wxHtmlWindow has its own coordinate and size values. When users click on a word on the browser window, the wxHtmlWindow will know which cell is being clicked by the coordinate values of mouse clicking position. In Vnet, we use the mouse right single click for the selection operation.

3.2.2.2 Obtain The Position of The Selection In The HTML Code

Vnet needs to know the position of the clicked word in the HTML source code. To obtain that, Vnet uses the coordinate information of wxHtmlCell it obtained as described in the previous section. When a webpage is presented in the browser window, Vnet creates a vector and a string. The vector contains all of the wxHtmlCells' coordinate values in turn. The string is a list of words which are contained by all the wxHtmlCells in the viewer. They are separated by line break characters in the string. Vnet knows the coordinate value of a selected cell. By looping though the vector we can find an index value for the selected cell. Since the text information of the web pages will not be changed, the index of each word in wxHtmlCell will not be changed either. The only thing that could be changed is the coordinate values of those wxHtmlCells when users resize the browser. Therefore, we renew the vector every time users resize the window or reload the page. With the index value, and the string which correspond to words in all the cells, we can obtain the unique position of the selection in the HTML source code.

3.2.2.3 Obtain A Sequence Of Tags That Represent The Position

With the index number and the text string from the previous step and the original HTML source code, Vnet can find a string of HTML tags that represent the position of the block of data which contains the selected words. Since Perl provides many functions and libraries for text string processing and HTML parsing, we used Perl to implement this search method. Using a utility class provided by Dr. Chou [3], we can easily call Perl programs from within a C++ program.

In Vnet, three variables are passed to the Perl search function, i.e., the HTML source code we get from our spider method, the text string which represents the words of all the

wxHtmlCells and the index of the selected wxHtmlCell. In the Perl function, the text string is split into an array TEXT_ARRAY by the line break characters. The original HTML code is parsed by the Perl package HTML::SPL [4, 5, 6]. In the loop that does the parsing, Vnet compares the text data in the original code and the text in the array TEXT_ARRAY at the same time it writes the structure tags into another array TAG_ARRAY. Since the text in TEXT_ARRAY originally comes from the HTML source code, if we ignore the spaces and line breaks, they should exactly match with each other. During the loop each time we find a structure tag we will update a variable START_CELL to the value of the current index number. Therefore, when the loop is stopped after reaching the selected cell index number, the START_CELL will contain the index number of the start wxHtmlCell for the group text.

When the index of the selected word is reached, Vnet stops the loop. It then starts another loop to find the end tag of the selected text and the index number of the end wxHtmlCell for the selected text. The loop will continually parse the HTML code from the stop point of the last loop and compare the text code with text in TEXT_ARRAY. When it reaches the first structure tag the loop is stopped. The tag is recorded in END_TAG and the current index number minus one is stored in the variable END_CELL. At the end we have an array of structure tags TAG_ARRAY and the end tag. With these information we can create a regular expression sequence that can query the original HTML code and extract the group of text that was selected by users, for example, a paragraph, a table cell or a block of text. The START_CELL and END_CELL are used to identify if a user clicked wxHtmlCell is selected or not during selection cancellations.

3.2.2.4 Generate Regular Expression Sequence From The Sequence of Tags

Although the information in TAG_ARRAY can be used to create a correct regular expression sequence, it contains too much information. It is possible to simplify the TAG_ARRAY. For example, if we want to extract the second paragraph in the HTML code showed in Graph 1, after the prior step we can create a regular expression sequence that contains all those structure tags:

```
.*?<html.*?< body.*?<h1.*?</h1.*?<p.*?</p.*?<p.*?>(.*?)</p
```

We can simplify it to

```
.*?<p.*?</p.*?<p.*?>(.*?)</p
```

This simplified regular expression sequence can also extract the second paragraph from the HTML code. Simplification is very important especially for complicated web pages containing many structure tags which are unessential in identifying user selections.

There are several methods to simplify the regular expression sequences.

1) Single tags:

This method starts from the last tag in array TAG_ARRAY, reads through the array from the tail to the head and collects the tags that are the same as the tail tag in TAG_ARRAY. This is similar to the simplification presented in the above example. This method is very efficient when the HTML code structures are simple. However, for complicated web pages, it does not provide a good simplification. For example, if a web page contains several big tables and a table cell in the last table is selected, the tag <td> will still appear too many times in the regular expression after the simplification because the method does not make the regular expression much shorter.

2) Hierarchical tree structure:

Most web pages that follow the HTML standard will have a tree structure for their HTML code which is organized by tags. Generally, the tag <html> will be the root. The following HTML code is an example:

```
<html>
    <head>
        <title>test page</title>
    </head>
    <body>
        <p>
            this is the paragraph one. this is the paragraph one.
            this is the paragraph one. this is the paragraph one.
            this is the paragraph one. this is the paragraph one.
        </p>
        <div>
            <p>
                this is the paragraph two. this is the paragraph two.
                this is the paragraph two. this is the paragraph two.
                this is the paragraph two. this is the paragraph two.
            </p>
            <p>test_1</p><p>test_2</p><p>test_3</p>
        </div>
        <span>
            <p>
                this is paragraph three. this is paragraph three.
                    <span>
                        this is inside paragraph three.
                    </span>
                this is paragraph three. this is paragraph three.
            </p>
        </span>
        <p>
            this is paragraph four. this is paragraph four.
            this is paragraph four. this is paragraph four.
            this is paragraph four. this is paragraph four.
        </p>
    </body>
</html>
```

If the fourth paragraph is selected, Vnet will generate the following regular expression sequence before simplification:

```
<html.*?<head.*?<p.*?</p.*?<div.*?<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?</div.*?<span.*?<p.*?<span
.*?</span.*?</p.*?</span.*?<p.*?>(.*?)</p
```

If we use method 1 to simplify this regular expression sequence, we will get:

```
<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?<p.*?</p.*?<p.*?>(.*?)</p
```

In the HTML code, if the "`test_i`" after paragraph two is increased to `test_n` in general, then we will have more than n pairs of `<p.*?</p.*?` in our regular expression. However, if we use

method 2 to simplify it, we can get the following regular expression sequence disregard the potential of n pairs of `<p.*?</p.*?`:

```
<html.*?<body.*?<p.*?</p.*?<div.*?</div.*?<span.*?<span.*?</span.*?</span.*?<p.*?>(.*?)</p
```

The following description details method 2.

Loop through tags in TAG_ARRAY from end to head. The loop contains following oppressions:

1) If the current tag is a start tag, leave it in a new array and go to the next tag; or

2) If the current tag is an end tag, run SIMPLIFY algorithm below till the tag exactly matches the start tag of the current end tag. This start tag then becomes the current tag. Go to next tag.

SIMPLIFY does the following things:

1) Find the exactly matching start tag for the current end tag, and find all such tags in-between them,

2) Put all detected tags into a new array. Ignore all the other tags in-between them.

3.2.2.5 Obtain The Selected Text Group

This step is simpler. By applying the regular expression sequence on the HTML source code, we can get the group of text we expected. However, the text string extracted directly from the source code could still contain some tags which are not included in table_1, so we need to let the string go through the parser again to remove those tags.

3.2.2.6 Mark The Text Group on The Browser

It is important to notify users what they have selected after they clicked the mouse. The most straightforward way is to highlight the text group they selected on the browser. To create

this visual function, we used the property of wxHtmlWindow which allows users to add self-defined tag handlers.

We created two tag handlers: <highlightstart> and <highlightend>. The first one sets the background color to yellow and the second one stops the color change and reloads the original color settings.

After the work described in Section 3.2.2.5, Vnet will insert <highlightstart> and <highlightend> to the HTML code by using the regular expression which we obtained. For example, if we get a regular expression sequence that will select the first paragraph of a webpage, it may look like this: <html.*?<body.*?<p.*?>(.*?)</p. We can modify it a little bit to create two more regular expressions. The first one, (<html.*?<body.*?<p.*?>), will be used to select HTML code before the first paragraph. The second one, <html.*?<body.*?<p.*?>.*?(</p.*?)$, will be used to select the HTML code right after the first paragraph. By inserting the new tag pair right before and after the code of the paragraph, and combining the three parts of code together again, we will have an HTML code that will highlight the user's selection when displayed in the Vnet browser. This process is used to highlight the HTML code for a single selection. For the table selections, it is different, and will be described in the next section.

At the end of these steps, the Perl function will return five values: the regular expression sequence, the selected text string, the modified HTML code (with highlights), and the start and end index values of the selected wxHtmlCells. The first, second, fourth and fifth values will be used to create the class that holds the selection's information.

3.2.3 Methods for Table Selection

Even though users are able to select a group of text now, if the information users want to select are contained in the columns of a big table with lots of rows, it is still very painful to select all of them individually. Furthermore, if the table is a dynamic table generated by some web server scripts, it is impossible to create a dynamic regular expression by selecting a fixed number of cells from the table. To solve this problem we created special methods for table selection.

These methods are created based on the property of some regular expressions that can do multiple selections. For example, if we want to select the information in the second column of the first table, we can use the following regular expression

“.*?<html.*?<body.*?<table.*?>(.*?)</table”

to extract the HTML code of the first table. On this table of the HTML code, we can use the regular expression <tr.*?<td.*?</td.*?<td.*?>(.*?)</td with option “m” to repeatedly extract information from the second table column on each row. In the same way, if users want the information in the first and second columns of the table, we can use the regular expression <tr.*?<td.*?>(.*?)</td.*?<td.*?>(.*?)</td and again option “m” to extract the information from the table. Vnet takes the right mouse button shift-click as the users’ intention of the special table selections.

To implement these methods we have to do the following additional steps of work:

1) Find which wxHtmlCell is selected;

2) Obtain the position of the selection in the HTML code;

3) Obtain a sequence of tags that represent the table position and the selected column number;

4) Obtain the groups of text from the HTML code; and

5) Highlight the selected groups of text.

Steps 1 and 2 are same as the steps described for single selection. The difference starts from step 3 and beyond.

3.2.3.1 Obtain A Sequence of Tags

This method is still implemented by Perl. Similar to single selections, we have three input variables which are the HTML source code, the wxHtmlCell text string and the index number of the selected wxHtmlCell. However, this method writes only <table> and </table> tags to the TAG_ARRAY. It also adds a condition in the first parsing loop determining if the tag is <td>. In that case the variable COL_NUM will be increased by 1. If the tag </tr> or </table> is reached, COL_NUM will be reset to 0. When it reached the selected cell index number and the first parsing loop stopped, COL_NUM will then contain the number of columns selected by users. The second parsing loop is also similar to the single selection case, but the difference is that this method tries to find the first <table> or </table> after the stop point of the first loop.

3.2.3.2 Obtain The Selected Text Groups And Highlight Them in The Browser

Since TAG_ARRAY contains only table tags, the simplification step as in the single selection is omitted. Steps 4 and 5 are finished in the same loop. First, with the method we used in step 6 of single selection and the regular expression we got from the last step, we can generate two more regular expressions. With these regular expressions, we can divide the original HTML code into three pieces:

1. HEADER, from the beginning to the selected table's start tag;
2. TABLE, the HTML code for the selected table which contains the selected columns; and
3. TAIL, from the selected table's end tag to the end of the HTML code.

We use a parsing loop to parse the second piece of HTML code for the tables. When there are the text strings in the tables, they will be compared with the text strings within TEXT_ARRAY. The index number starts at START_CELL that we got from last step. A counter is used to count the number of <td> tags seen. Each time when counter value is equal to COL_NUM, the following actions will be taken:

1) A <highlightstart> tag will be inserted into the HTML code;

2) The text code will be added to a text string SELECT_STR; and

3) The current index number will be put into a variable START.

After that, when the first </td> is reached, the following actions will happen:

1) The <highlightend> will be inserted into the HTML code;

2) The adding of the text to SELECT_STR will be stopped;

3) A line break mark, "\n", will be added at the end of SELECT_STR;

4) The current index number will be written into variable END; and

5) The values of START and END are pushed into array PAIRS.

Each time a </tr> is reached, the counter will be reset to 0.

After this looping, we will have the following results:

1) A segment of HTML code with selected text highlighted;

2) SELECT_STR with all its text information in the selected column; and

3) PAIRS that have the starting and ending index numbers of all the pieces of selected wxHtmlCells. This information is useful to identify whether or not a wxHtmlCell is selected.

At the end, we combine the HEADER, a fragment of table HTML code we obtain from the loop, and TAIL. This operation generates new HTML source code that highlights table cells. The following information will be returned to the main program:

1) A regular expression for extracting the selected table,

2) SELECT_STR,

3) PAIRS,

4) START_CELL and END_CELL, and

5) New HTML code,

3.2.4 Data Structure to Hold Selections

We create a class called HtmlSelection to hold the information from these two types of selections. The class contains a Boolean value IS_TABLE to designate whether it is a single selection or a table selection. If a single selection, IS_TALBE will be set to false, otherwise it will be set to true. The class will store the regular expression sequence, the selected text string, and the starting and ending index numbers that represent the index range for the wxHtmlCells of the selected text group. If a table selection, IS_TABLE will be set to true, otherwise it will be set to false. The following information will be stored in the class: a regular expression for the selected table, strings that hold the text information of the selected columns, the selected column numbers, information in PAIRS which represents the index ranges of text in the selected columns, and the starting and ending index numbers for the wxHtmlCells of the selected table, representing its index range.

3.2.5 Insertion of Selections

All selections are stored in a Vector HtmlSelection. In Vnet, the order of selections is decided by the order of appearance of the selected text in the original HTML code. Since we have the value of the index number of all words in the HTML code as well as the index range of the selected text, we can easily determine the order of the selections if they are all single selections. However the table selections will be more difficult to be determine because a table

selection has a series of selected strings combined with a series of index ranges. It is difficult to decide the order between a table selection and a single selection if the index range of the single selection appears within the index ranges of the table selection. To avoid this problem, Vnet decides that single selections are always located before the table selection if they belong to the same table.

We now can state the rule for setting order, and the methods for inserting new selections follow this rule. If the new selection is a single selection, the insertion conforms to the following operations:

Loop through the selection vector using the following processes:

1) if the current selection is a single selection and has an index range value smaller than the new selection, do nothing and continue;

2) if the current selection is a single selection and has an index range value greater than the new selection, insert the new selection before the current selection and terminate the loop;

3) if the current selection is a table selection and has a selected table index range smaller than that of the new selection, do nothing and continue the loop;

4) if the current selection is a table selection and has selected table index range greater than the new selection, insert the new selection before the current selection and terminate the loop; and

5) if the current selection is a table selection and the new selection has an index range within the table index range of the current selection, insert the new selection before the current selection and terminate the loop.

The case in which insertion of a new selection which is a cell selected by a table selection will not exist because that would lead to a cancellation of that table selection.

If the new selection is a table selection, the insertion is made as follows:

Loop through the selection vector using the following processes:

1) if the current selection is a single selection and has an index range value smaller than the table index range of the new selection, do nothing and continue;

2) if the current selection is a single selection and has an index range value bigger than the table index range of the new selection, insert the new selection before the current selection and terminate the loop;

3) if the current selection is a single selection and has an index range value within the table index range of the new selection, test whether the current selection's index range is within one of the index ranges of the selected column for the new selection. If the answer is yes, remove the current selection and continue. If the answer is no, do nothing and continue;

4) if the current selection is a table selection and has a table index range value smaller than the table index range of the new selection, do nothing and continue;

5) if the current selection is a table selection and has a table index range value greater than the table index range of the new selection, insert the new selection before the current selection and terminate the loop; and

6) if the current selection is a table selection and has a table index range value equal to the new table index range values, then another column in the same table as the , current selection is selected. Then the information in the new selection, including the column number, the selected string of the column, and index ranges of wxHtmlCells of the column will be added to the current selection, following which the loop will be terminated.

3.2.6 Methods for Canceling Selections

To cancel selections, users can either clear all selections or only cancel some (presumably mistaken) selections. If users want to clear all selections, they can click a "Clear" button. Vnet will clear all the elements in the selection vector, clear all texts in the output view panel, and

remove all highlights on the browser. Users will then know that all selections have been removed.

3.2.6.1 Canceling Selections Individually

When users singly right-click wxHtmlCells on the Vnet browser, Vnet will do the following things:

1) Loop through the selection vector and

a. If the current selection is a single selection and the index of the currently selected wxHtmlCell is within the index range of the current selection, remove the current selection from the vector, remove the highlight tags for this selection, and terminate the loop. Details as to how to remove the highlight tags will be described in the next section;

b. If the current selection is a table selection and the index of the currently selected wxHtmlCell is within the index ranges of one of the selected columns in the current selection, remove the column information from that table selection. The information removed includes the column number, the selected string for that column, and the index ranges of the selected text for that column. After that, if the table selection is empty, e.g., if all the selected columns are removed, remove the selection from the vector. The highlight tags for this selection must also be removed and the loop terminated; and

c. For all other cases, do nothing and continue.

2) If the loop finishes normally, it means that the user has clicked in an area that has not been selected yet, so proceed to the selection process.

3.2.6.2 Removing Highlight Tags

To cancel selections, we must not only remove the HtmlSelections from the vector, but we must also remove the related highlights from the HTML code so that users can observe the

results of the cancellation. Since we have a regular expression for these selections, we can use this advantage to remove the selection, and we will use Perl to implement this function.

The input variables to Perl function enclose the HTML code, the regular expression sequence and an integer that represents whether the selection is a single selection or a table selection. If it is a single selection, we set the integer to 0. Otherwise, we set the value to be the column number of that selection. The returned value will be the updated HTML source code.

If the selection is a single selection, it is simple to remove the highlight tags using a process similar to that of adding highlight tags. We create two more regular expression sequences based on the regular expression we have. Through these three sequences, we query the HTML source code and obtain the following three pieces of HTML code:

1) HEAD, HTML code extending from the beginning to the position just before the tag <highlightstart>;

2) BODY, code we select by the original regular expression sequence, containing <highlightstart> and <highlightend> tags; and

3) TAIL, HTML code extending from the position after the tag <highlightend> to the end of the file.

We can just remove the highlight tags in BODY to create a new BODY, and then combine HEAD, new BODY and TAIL together to create and return a new HTML source code segment.

If the selection is a table selection, the method to remove highlight tags is also similar to the method of adding highlight tags.

With the regular expression for the selected table we create two more regular expressions and divide the HTML source code into the following three segments:

1) HEAD, HTML code extending from the beginning to the selected table’s start tag;

2) TABLE, HTML code for the selected table containing the selected column; and

3) TAIL, HTML code extending from the selected table's end tag to the end of the file.

Now we can create a parsing loop to parse the HTML code in the TABLE as follows. Vnet has a 0-initialized counter to count the number of <td>s read. In the loop Vnet will remove the highlight tags if the counter is equal to the column number. Each time, if a </tr> is reached, set counter to 0.

Now we have a new TABLE with the appropriate highlights removed. By combining the HEAD, new TABLE, and TAIL together, we obtain and return a new HTML source code fragment.

3.3. Generating the Output View

Users can preview what they have selected on the output panel. After they preview data and are satisfied with them, they can save the data to a file. Alternatively they can clear the selection and go back to the previous steps. The clearing process is similar to that described in section 3.4.2, but it also must remove the text on the output panel. This section will describe how the output text is generated.

The output string results from combining all the selected strings in the selection vector. For a single selection, we will just use the selected string directly. For a table selection, if it contains more than one selected column, we will merge the selected strings for those columns. After the merge, the format of the final selected string for these columns looks like this: text strings from each row are separated by line breaks and text strings from each cell are separated by tab breaks. The original line breaks and tab breaks in the HTML code are removed when they are stored in the table selection.

3.4 Generating Perl Code

After users save the data on the output panel to a file, they can terminate Vnet, since they get what they want immediately. However, most users would like to avoid the effort of reading through web pages again to find desired information in subsequent uses of the system. They may need to go to the same web sites and look for information that will always be put into the same position. Vnet can create Perl programs to help users find this information.

Vnet can help a user create a Perl program according to what he or she has done from the spider search to the selection. By clicking the create Perl program button, the Perl code will be output to the Perl panel. Users that have some Perl programming skills can modify the code if they wish. Users have the option of running the Perl code by clicking the "Run" button to test whether the produced result is the same as that shown on the output panel.

Initially, the program will create a spider search engine for the Perl program. The search engine will retain the same structure as that used in Vnet, with the parameters the users have input to the spider search engine retained as default values. Users will be able to specify the new values if they wish.

The second part of the Perl program is for data extraction. The required regular expression information comes from the selection vector. However, we still must simplify the regular expression sequences from single selections.

3.4.1 Combine The Regular Expressions From Single Selections

To avoid messing up the order of the selections, we combine the regular expression sequences only from neighboring single selections. We will use an example to show how two regular expression sequences may be combined. Suppose we have two regular expression sequences: `<html.*?<body.*?<p.*?>(.*?)</p` and

`<html.*?<body.*?<p.*?</p.*?<p.*?>(.*?)</p`. The first sequence will select the first paragraph from HTML code and the second will select the second paragraph. We can split these two regular expression sequences into two arrays using string "`.*?`", and then begin comparing two arrays using a loop.

During the loop, push the same elements into a new array. For elements present in the first one but not in the second, insert the remaining elements in the first array into the new array, and then insert the remaining elements in the second array into the new array. After the loop, we combine the new array into the string with "`.*?`" between each pair of elements. This produces a new regular expression sequence: `<html.*?<body.*?<p.*?>(.*?)</p.*?<p.*?>(.*?)</p`. that can extract both first and second paragraphs from the HTML source code.

3.4.2 Completing the Regular Expression for the Table Selection

To make the cancellation of the table selection simpler, we do not create regular expressions for the selected columns in the table selection. To create regular expressions for them, we need only the selected column numbers. For example, if we have 1, 3 and 4 as the selected column number we can create a regular expression sequence for them as follows:

```
<tr.*?<td.*?>(.*?)</td.*?<td.*?</td.*?<td.*?>(.*?)</td.*?<td.*?>(.*?)</td
```

We need the following two steps to produce each table selection in the Perl program

1) Use the regular expression sequence for the selected table to extract HTML code for that table; and

2) Use the regular expression sequence we just created to extract text in the selected columns.

CHAPTER 4. REAL-WORLD EXAMPLE

We applied Vnet to the task which is detecting information about an Enzyme with E.C. (Enzyme Commission) number 5.3.1.1 on KEGG's web site. KEGG (Kyoto Encyclopedia of Genes and Genomes) is an online bioinformatics database provided by Kanehisa Laboratories in Japan. Their pathway database is one of the most popular pathway databases in the world. Their web address is http://www.genome.jp/kegg/. Vnet can browse web pages from this address.

4.1 Web Browsing and Spider Searching

There are two ways to search information using Vnet. One is to search the information manually. Inputting a web address into a text field at the top of the browser window and hitting the Go button will cause Vnet to display that web page in the same manner as other web browsers. Users can then search for desired information by viewing the web pages or clicking on the links. Figure 6 is a screen snapshot of a KEGG web page produced by Vnet.

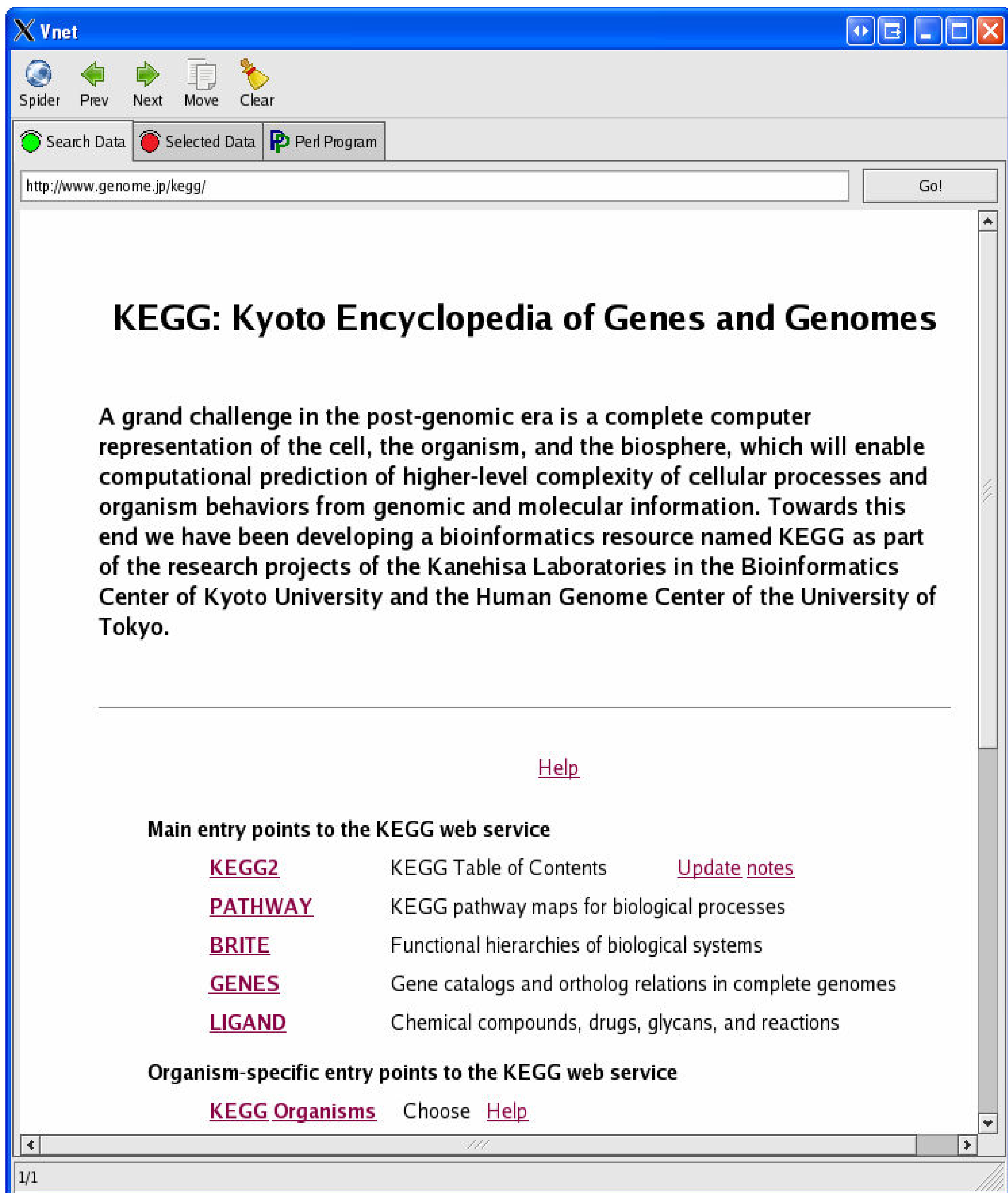

Figure 6: Snapshot for Vnet Search Data panel.

The other search method is to use spider search engine in Vnet. Like shown in Figure 7, if the Spider button on the top of the window is clicked, a window will pop up. This window is provided for users to specify the factors related to the spider search, i.e., a URL, keywords,

and a level. The URL is a web address used as the root during the spider search, keywords are the words expected to be found on the target web pages, and the level is the depth into which users want to dig at that web site. Generally, we recommend a maximum level of 3 at most. The running time of the spider could be long, depending on many factors such as Internet speed, size of the web pages, number of links on a given web page, and so on. If the level number is too large, the running time of the spider could be extremely long.

Therefore, to save users' time and make the search more efficient, we suggest combining two search methods. First, find an URL and start to search manually from it to some level with Vnet browser. Second, copy the URL from the browser URL field to the spider's pop up window; from that web page users can use the spider to find desired information. Returned pages will be put into a vector, and users can click the "next" or "previous" button to examine them to find the desired information. In this example users would browse the pathway page. Since we know that the pathway of Inositol metabolism uses the selected enzyme, we click the link to the Inositol metabolism pathway map and then start the spider. The URL of the pathway map is shown in a URL popup window. We just specify the keyword "5.3.1.1" and level 1. Figure 7 shows the operation at this step. After a search delay, one of the returned pages is shown on the browser, and the number "1/5" appears on the status bar. That means that 5 pages have been found and the displayed web page is the first one. By clicking the "next" button users can find the page containing information about enzyme 5.3.1.1.

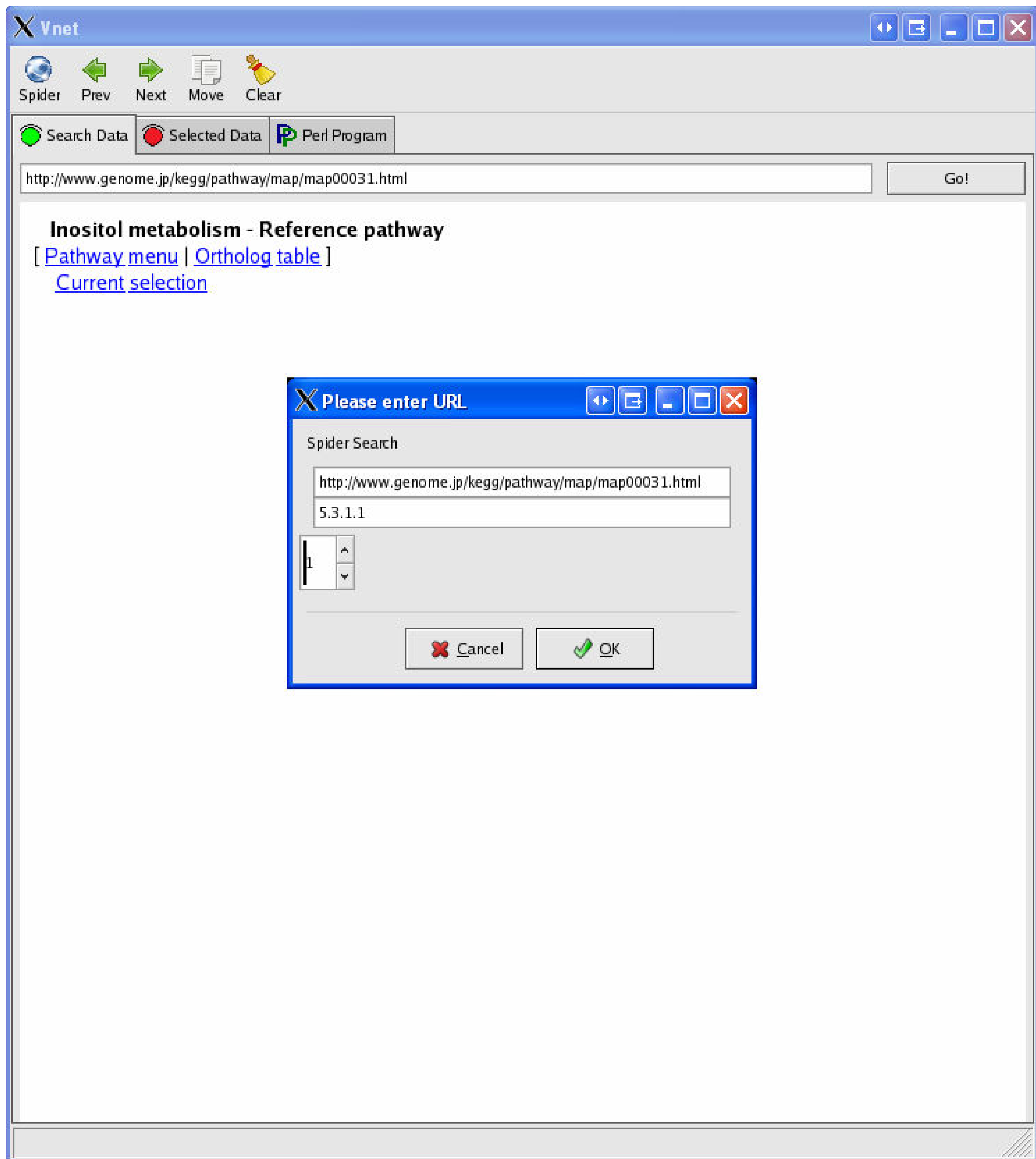

Figure 7: Snapshot for using spider search engine in Vnet.

4.2 Data Selection.

In data selection wee want to find information about name, class, sysname and references. If we right-click on the cells containing information we want, selected information in the table is highlighted. Figure 8 shows the selected data in the browser.

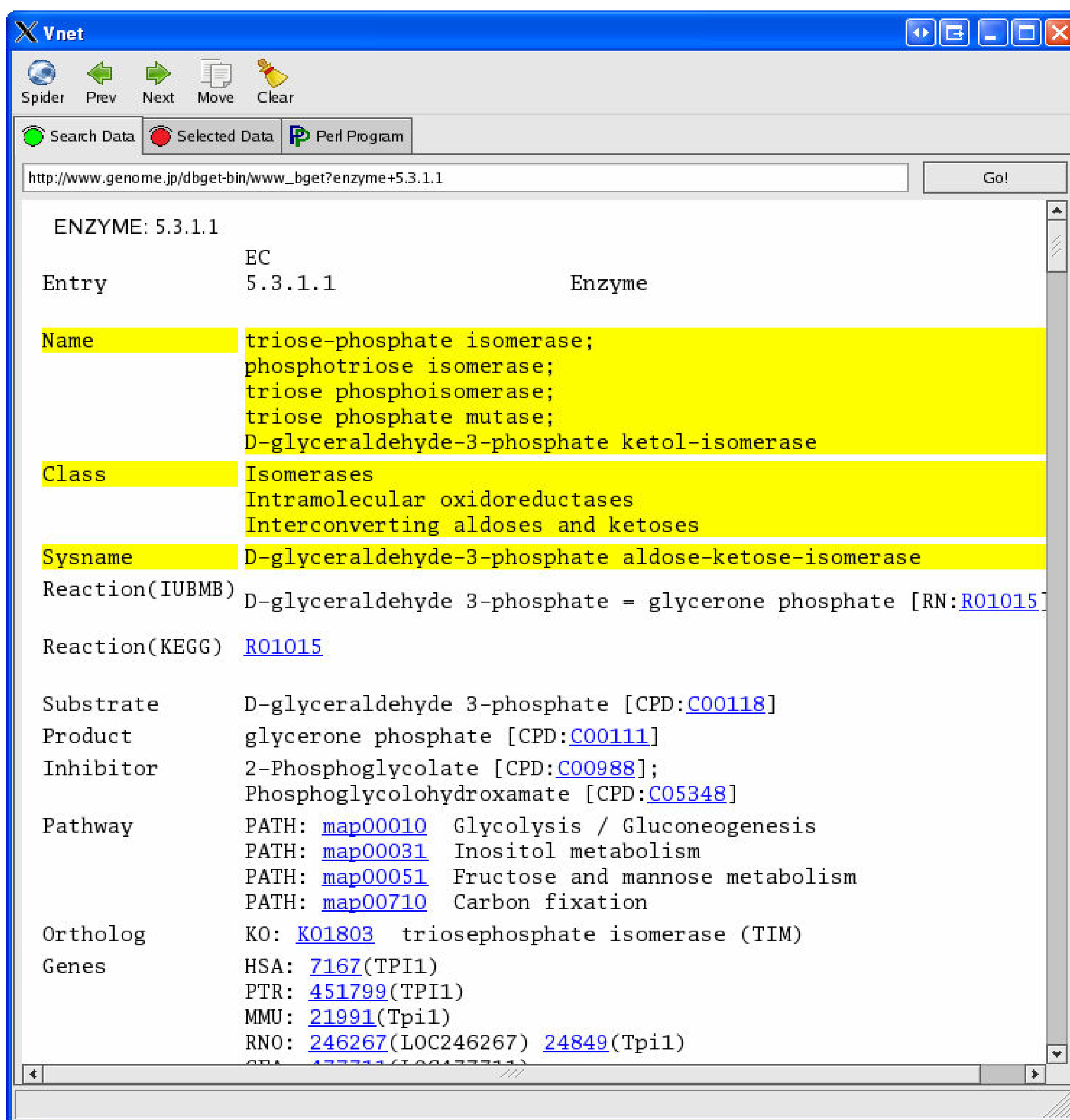

Figure 8: Selections on Vnet Search Data panel.

4.3 Output Data

When users finish the selection described in the previous section, they can click the "move" button to move the selected data to the output panel, as shown in Figure 9.

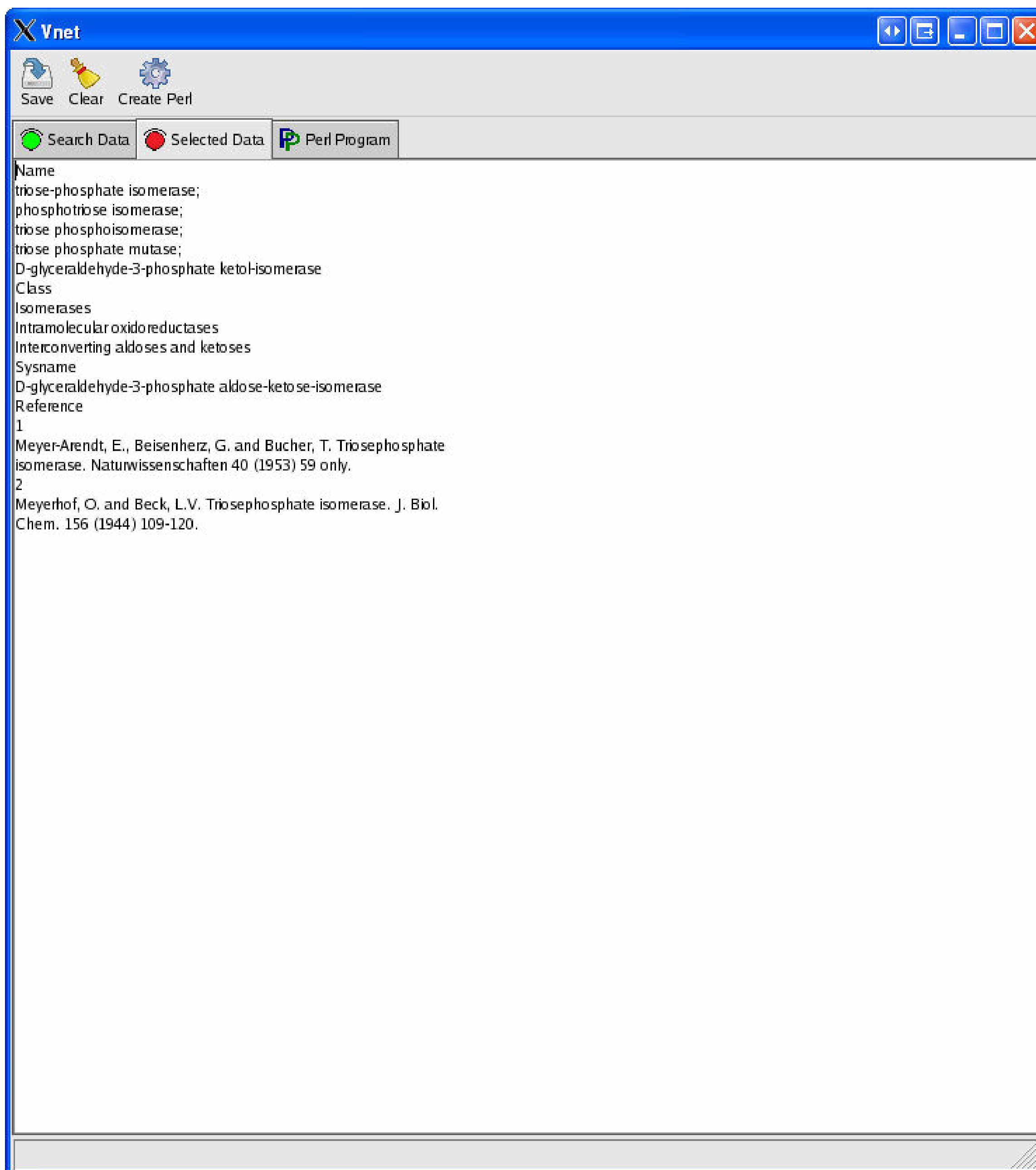

Figure 9: Output data view on Selected Date panel.

4.4 Generate a Perl Code

If users are satisfied with the data selected, they can generate a Perl program which reflects the information contained in the selection. If users click the "Create Perl" button on the output panel, Perl code will be generated on the Perl panel. Users can test the Perl code by clicking the run button on the Perl panel. The result will be shown on a popup window like Figure 10 shown below.

Figure 10: Snapshot for running generated Perl program on Vnet.

CHAPTER 5. CONCLUSION AND FUTURE WORK

In this thesis, we have discussed the need for automatically and easily obtaining the most current data on selected web pages. We have also described limitations of existing software and tools that can be used to do part of this type of searching and extraction. We introduce Vnet, software developed under this thesis work, as an improvement in the methodology for searching web pages, for selecting specific data in particular areas on web pages, and for generating a Perl program to automate the whole process. Finally, a real-world example is introduced to evaluate the functionality of Vnet. The results produced by the example reveal that Vnet greatly enhances efficiency in doing this kind of work.

The main contribution of this thesis work is the development of Vnet for Internet data searching and extraction. While these processes can be done separately using existing tools and software, this is usually both tedious and difficult for the end user. Vnet solves this problem by combining those functions into one program and generating a Perl program to perform the whole process in one run. By considering the real world example and the way Vnet solves the problem, the improved efficiency and convenience of using Vnet are obvious.

There are several ways that Vnet can be enhanced in the future. Some popular web sites provide their own search engines. For those that don't, they still can index their websites with Google and support a customized Google search. Under various conditions, users can often take advantages of the web formats on those websites and obtain even more related information. Currently, Vnet does not yet have a property to support auto-filling a web form. In fact it can not even show a web form on the browser because wxHtmlWindow does not provide this property. Therefore, the property of displaying the web form on the browser also must be implemented before the property of web form auto-filling can be added.

The ability to modify output would be another useful addition. For example adding special symbols at the beginning of selections of interest to mark their differences would be helpful in later dealing with the various selected data.

Currently, Vnet saves only the regular expressions of users' selections into Perl. Therefore, if users want to modify selections after they terminate the Vnet, they must start over again with Vnet. If Vnet could save searching and extraction information to files that Vnet itself could import, users could modify the selections by loading a special file to Vnet and working on it.

REFERENCES

[1] Peter Christian *Web Publishing for Genealogy* Genealogical Publishing Com, 2000.

[2] Bernard J. Jansen, Amanda Spink *Public Searching of the Web* Springer, 2004.

[3] Hui-Hsien Chou VECT: an automatic visual Perl programming tool for nonprogrammers, *BioTechniques* 38: 615-621 April 2005.

[4] Sean M. Burke, *Perl & LWP*, O'REILLY, 2002.

[5] Tom Christiansen & Nathan Torkington, *Perl Cookbook*, O'REILLY, 1998.

[6] Randal L. Schwartz, Tom Phoenix, brian d foy, *Learning Perl*, O'REILLY, 2005.

[7] da Silva, A.S. Veloso, E.A. Golgher, P.B, CoBWeb-a crawler for the Brazilian Web, *String Processing and Information Retrieval Symposium, 1999 and International Workshop on Groupware*, pp. 184-191, 1999.

[8] Sang Ho Lee, Sung Jin Kim, and Seok Hoo Hong, On URL Normalization, *ICCSA 2005*, LNCS 3481, pp.1076-1085, 2005.

[9] Julian Smart, Kevin Hock, Stefan Csomor *Cross-Platform Gui Programming With WxWidgets* Prentice Hall PTR, 2006.

[10] Jeffrey E. F. Friedl *Mastering Regular Expressions* O'REILLY, 2006.

[11] Jennifer Niederst Robbins *HTML Pocket Reference* O'REILLY, 2002.

[12] Haruo Hosoya, Benjamin C. Pierce Regular expression pattern matching for XML *Journal of Functional Programming (2003), 13: 961-1004.*

ACKNOWLEDGEMENT

I am grateful to my major professor, Dr. Hui-Hsien Chou for his guidance, patience, mentoring, and support throughout my graduate study.

I would also like to thanks Dr. Gadia and Dr. Gu to serve on my committee and offer their help at any time.

This work was supported in part by National Institutes of Health Grant 4R33GM066400. I want to thank NIH for providing the funding of this research.

Last but certainly not least, I am forever indebted to the love and support of my family and my wife.