Web Page Downloading and Classification

Loc Q. Tran, Chan W. Moon, Daniel X. Le, George R. Thoma
National Library of Medicine
8600 Rockville Pike, Bethesda, MD 20894
loc_tran@nih.gov

Abstract

This paper describes the processes of downloading and classifying Web-based articles in on-line medical journals as a preliminary step to extracting bibliographic data to populate MEDLINE®, the widely used database of the National Library of Medicine (NLM). The processes are combined to develop an automated system named "Web Page Downloading and Classification". The system downloads the Web pages using Microsoft's Windows Internet API tool called WinInet, and a combination of several Artificial Intelligence (AI) techniques including the Breadth-First search algorithm and the Constraint Satisfaction method. The Breadth-First search algorithm and the Constraint Satisfaction method are then used to traverse the Web page's links, identify these pages as abstract, full text, PDF or image files, recognize and generate the successors of the downloading pages.

1. Introduction and background

One of the achievements in the area of document imaging at NLM is the successful development and deployment of the Medical Article Record System (MARS), which is used to automatically extract citation fields from scanned medical journals [1, 2]. The MARS system produces over one-third of the medical citation records for inclusion in MEDLINE database, the premier database at NLM used worldwide.

During the past few years, the World Wide Web has become an important source for information, and biomedical journal publishers have begun to publish articles on the Internet. To take advantage of these on-line journals, the Lister Hill National Center for Biomedical Communications, a research and development division of NLM, is developing an automated system temporarily code-named WebMARS for *Web-Based Medical Article Record System* to create citation records for the MEDLINE database from on-line journals. The system downloads and classifies Web document articles, parses and labels the article contents, extracts and reformats the citation information from the article, presents the entire citation to operators for reconciling (validation), and uploads the citation records to the MEDLINE database. This paper describes one component of this ongoing effort at NLM: *the Web Page Downloading and Classification.*

The system consists of two processes: *Downloading* and *Classification*. The first is based on WinInet software tool and a combination of the Breadth-First search algorithm and the Constraint Satisfaction method to traverse the Web page's links, recognize and generate the successors of the downloading pages. The second relies on the contents of the hyperlinks and uses Constraint Satisfaction method to classify Web-based files as abstract, full text, or PDF files.

Proceedings of the Fourteenth IEEE Symposium on Computer-Based Medical Systems (CBMS'01)
1063-7125/01 $10.00 © 2001 **IEEE**

2. System design

Of the two processes in this system, the *Downloading* process uses Microsoft's Windows Internet API tool (WinInet) to connect to the Web servers, and to download the journal citation data in several formats: HTML, images, and portable document format (PDF). The tool is used to send requests to the Web server to download the pages, determine a transfer mode (ASCII or binary) based on the Web page's header, and to handle errors returned from the Web server. The Breadth-First search algorithm is applied for this process to control data flow and keep track of the progress. The three lists: *Open, Closed* and *Revisited* are used in the search algorithm to make sure the *Downloading* process is moving smoothly, the same Web page is not downloaded twice, and unsuccessfully downloaded page are revisited for a second try. The *Classification* process will classify the Web pages and the types of their contents (abstract, full text or PDF). The Constraint Satisfaction method is used to fulfill this goal, ideally based on the hypertext, and displayed text to classify the successors. This method also uses the contents of hypertext to identify the set of constraints. The *Classification* is accomplished during the *Downloading* process, so only the necessary pages will be downloaded and placed in the correct directories according to their types.

2.1. Downloading process using the Breadth-First search algorithm

Since the links among Web pages are similar to a tree structure, the Breadth-First search algorithm is chosen to control data flows and traverse the tree for the *Downloading* process.

The Breadth-First search is implemented using two lists - *Open* list and *Closed* list - to keep track of the progress through the state space. *Open* list is maintained as a queue, first-in-first-out (FIFO). It contains all states that have been generated but whose children have not been examined. The order in which states are removed from *Open* list and recorded in *Closed* list determines the order of the search. When the search is finished, the *Closed* list contains the path of states that have been examined through the search process.

In addition to establishing a search direction in this Web page downloading system (by the hypertext links that start with HREF=), the Breadth-First search algorithm determines the order in which states are examined in the tree. It explores the space level-by-level. Only when there are no more states to be explored at a given level, does the algorithm move on to the next level [3]. At the tree of the hyperlinks, the Breadth-First search algorithm considers the states in order from A to Z, as shown in Figure 1.

For this system, we implement the downloading process using Breadth-First search with three lists: *Open*, *Closed*, and *Revisited* to keep track of the progress throughout the downloading process. Each list consists of a series of nodes, which contain the uniform resource locator (URL) addresses of the Web pages. The lists are defined as follows:

- *Open list*: stores the addresses of pages waiting to be downloaded.
- *Closed list*: stores the addresses of pages that are successfully downloaded.
- *Revisited list*: stores the addresses of pages that failed during the downloading process, and are to be revisited later.

```
//Start algorithm
begin
   Open := [ S0 ];  // set flag(1) to  S0
   Closed := [];
   Revisited := [];
```

```
    while Open != []
    begin
      pick the head node in Open list, call it X;
      if download X succeeds
         generate children of X with flag(1);
         add those children to the tail of Open;
         remove X from Open;
         put X on Closed;
      else //download X fails
         remove X from Open;
         if node has flag(1)
            add X to the head of Revisited;
         else //node has flag(0)
            add X to the tail of Revisited;
         end if
      end if
      if Open = [] && Revisited != []
         if the head node of Revisited has flag(1)
            remove the node from Revisited;
            set flag(0) to the node;
            put it on Open;
         end if
      end if
    end while
end
//End of algorithm
```

Beginning with the first journal issue page, the process sends a request to the Web server to make a connection and to download the page. During the download, the links of the current downloading page are classified (by the *Classification* process), and the children are generated as nodes with flag(1) and added to the tail of the *Open* list. The child pages that have already been discovered (they are already on either *Open*, *Closed*, or *Revisited* list) are eliminated. Then the successfully downloaded page is classified by type and saved to a corresponding directory. If the download is successful, the node of the successfully downloaded page is removed from *Open* list and recorded to the *Closed* list. Otherwise, it will be added to the head of the *Revisited* list for a second try. The *Open* list now contains the successors of the previous downloaded page, which are waiting to be downloaded. The downloading continues until the *Open* list is empty (*Open* = []), then the *Revisited* list will be examined. If the *Revisited* list is not empty (*Revisited* != []) and its head node has flag(1), the node's flag is changed to 0 and moved to the *Open* list. If the download of a Web page fails again at the second try, it will be added to the tail of the *Revisited* list for manually downloading later. The *Downloading* process continues until the *Open* list is empty, and *Revisited* list is empty or its head node has flag(0).

Trace of the Web page downloading process with 4-level pages and 14 satisfied links: (Figure 2)
We start at the first page of the journal issue:

1. *Open* = [S_0]; *Closed* = []; *Revisited* = []
2. *Open* = [S_1 , S_2]; *Closed* = [S_0]; *Revisited* = []
3. *Open* = [S_3 , S_4 , S_5 , S_6]; *Closed* = [S_0 , S_1 , S_2]; *Revisited* = []
4. *Open* = [S_7 , S_8 , S_9 , S_{10} , S_{11} , S_{12}]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6]; *Revisited* = []

5. *Open* = [S_8 , S_9 , S_{10} , S_{11} , S_{12}]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6]; *Revisited* = [S_7] (S_7 failed)
6. *Open* = [S_9 , S_{10} , S_{11} , S_{12}]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8]; *Revisited* = [S_7]
7. *Open* = [S_{10} , S_{11} , S_{12}]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8]; *Revisited* = [S_9 , S_7] (S_9 failed)
8. *Open* = []; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12}]; *Revisited* = [S_9 , S_7]

(*Open* = [] and *Revisited* != [], change flag(1) to flag(0) and move it from *Revisited* to *Open*)

9. *Open* = [S_9]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_9 , S_{10} , S_{11} , S_{12}]; *Revisited* = [S_7]
10. *Open* = [S_{13} , S_{14}]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9]; *Revisited* = [S_7]
11. *Open* = []; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9 , S_{13} , S_{14}]; *Revisited* = [S_7]

(*Open* = [] and *Revisited* != [], change flag(1) to flag(0) and move it from *Revisited* to *Open*)

12. *Open* = [S_7]; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9 , S_{13} , S_{14}]; *Revisited* = []
13. *Open* = []; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9 , S_{13} , S_{14} , S_7]; *Revisited* = []
14. *Open* = []; *Closed* = [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9 , S_{13} , S_{14} , S_7]; *Revisited* = []

Download ends when *Open* = [], and *Revisited* = [] or its first node has flag(0);

Path of states: [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_7 , S_8 , S_9 , S_{10} , S_{11} , S_{12} , S_{13} , S_{14}]

The order of downloading states: [S_0 , S_1 , S_2 , S_3 , S_4 , S_5 , S_6 , S_8 , S_{10} , S_{11} , S_{12} , S_9 , S_{13} , S_{14} , S_7]

2.2. Classification process using the Constraint Satisfaction method

After connecting to the desired publisher Web server, the system requires to traverse the Web page's hyperlinks to collect all articles of a particular issue. The hyperlinks embedded in a Web page are broad, unlimited and varied. They can be linked to the same page, to pages on the same Web server, or to any Web servers around the world. Hyperlinks may be associated with an image or a displayed text.

Since we are interested in Web journal articles only, hyperlinks and the displayed text of the hyperlinks will be analyzed and validated against a set of predefined constraints. If they are qualified, Web pages associated with them will be marked as potential pages to be downloaded. It is noted that we are limited to classified hyperlinks to the Web pages related to file types such as: abstract, full text, PDF and images. Since the contents of hyperlinks consist of useful information about file types such as {"abstract", "abs", "list", "[Abstract]", "Abstract"} for abstract, {"full", "Full Text", "Full text", "References"} for full text, {"PDF", "pdf"} for PDF, these can be used to determine the valid child links and the appropriate directory for the downloaded page. Moreover, the *Classification* process only chooses the links on the same server as the starting journal issue page. The *Classification* process has two important responsibilities:

- Classify the links for the *Downloading* process to generate successors of the currently downloading page.
- Classify the downloaded data to place it in the correct directory corresponding to its type.

For example: if there is an initial page called "parent.html", and the classification process found "file1.html" (abstract format)," file2.html " (full text format), and" file3.pdf" (pdf format). Then, the downloading process makes "file1.html", "file2.html" and "file3.pdf" children of "parent.html" file, and it continues building the tree with children of "file1.html" and "file2.html". However, "file3.pdf" does not have any children generated. Then "file1.html" will be stored under *Abstract* directory, "file2.html" under *FullText* directory, and "file3.pdf" under

PDF directory after they are downloaded (Figure 3). Other downloaded files that the *Classification* process missed or could not classify will be stored under the *Other* directory, and the subsequence system will classify them later.

We implement the *Classification* process using two levels of Constraint Satisfaction with the sets of constraint variables [4]. Call {U} a set of all the links that we found in the downloaded page. The first set of constraints, call it {V}, is the set of common links found in medical journal Web pages such as: {"home", "subscriptions", "archive", "mailto", "help", "login", "search", "feedback", "findex", "shtml", "lookup"}, etc., that we can eliminate at the first level of classification. {A} is a set of constraints for abstract. {F} is a set of constraints for full text. {P} is a set of constraints for PDF file. The potential successors will be: {S} = {U} – {V}

```
procedure LOCATION()
begin
    x ∈ {S};
    if x ∈ {P}
        store x under "PDF" folder;
    else
        if x ∈ {A}
            store x under "Abstract" folder;
        else if x ∈ {F}
            store x under "FullText" folder;
        else
            store x under "Other" folder;
        end if
        generate x's children for {U};
        call ELIMINATION to classify;
    end if
end LOCATION
```

```
procedure ELIMINATION()
begin
    for each x ∈ {U} do
        if x ∉ {V} && x on server then
            add x to {S};
        else
            delete x;
        end if
    end for
end ELIMINATION
```

During download, the addresses are classified at the second level of classification to determine the appropriate directory for the downloaded pages with these sets of constraints: (Figure 4)
{A} = {"abstract", "abs", "list", "[Abstract]", "Abstract"}
{F} = {"full", "Full Text", "Full text", "References"}
{P} = {"PDF", "pdf"}

3. Experimental result

We conducted an experiment using a T1 line on 28 journal Web sites. The average time to download a Web page is about 3.5 seconds. For a specific journal issue with 29 articles consisting of 82 files (9,295 Kbytes), the system takes approximately 4 – 5 minutes (32 – 38 Kbytes/second).

4. Summary

We have presented a Heuristic search with a simple Constraint Satisfaction, and a hyperlink-driven program for an automated Web journals downloading system. We also discussed the "back jumping" operation in the search algorithm using our recommended feature called "*Revisited*" to automatically re-download any Web pages that failed to download. Our preliminary results show that our system performs well on a small set of Web journals with

known page layouts. A future task is to expand the current system's configuration to learn and adapt to any type of Web page journal layouts.

5. References

[1] G.R. Thoma, "Automating Data Entry into MEDLINE", Proc. 1999 Symposium on Document Image understanding Technology (SDIUT 99), Annapolis, MD, April 1999, pp. 217-218.

[2] D. Le, J. Kim, G. Pearson, and G. Thoma, "Automated Labeling of Zones from Scanned Documents," Proc. 1999 Symposium on Document Image understanding Technology (SDIUT 99), Annapolis, MD, April 1999, pp. 219-226.

[3] G.F. Luger, W.A. Stubblefield, "Artificial Intelligence - Structure and Strategies for Complex Problem Solving", Second Edition, The Benjamin/Cummings Publishing Company, Inc., 1993.

[4] V. Kumar, "Algorithms for Constraint Satisfaction Problems: A Survey", AI Magazine, Spring 1992, pp. 32-41.

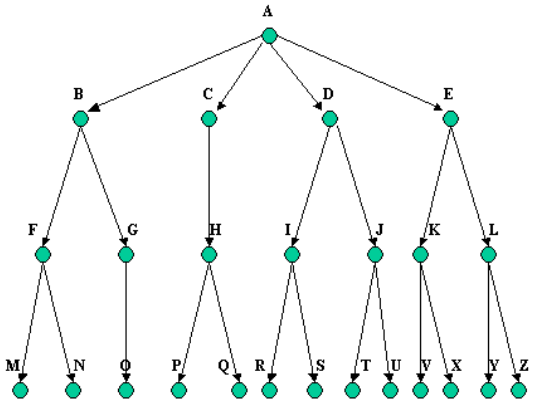

Figure 1. Hyperlink tree

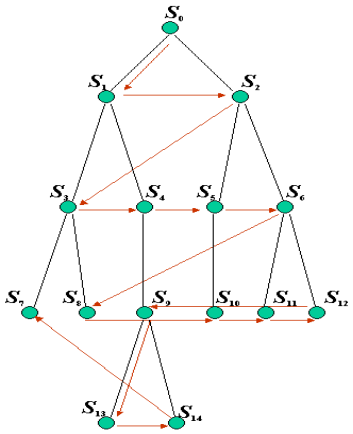

Figure 2. Downloading process

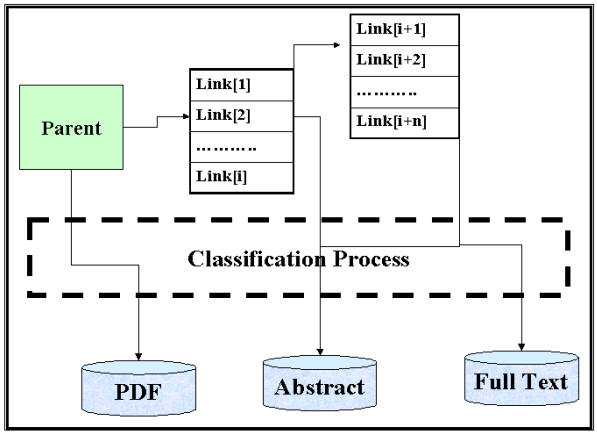

Figure 3. Classification process

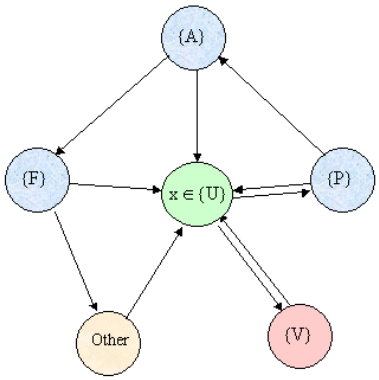

Figure 4. Constraint graph