Assignment 1: Build a Web Crawler

Mark Cline
Department of Computer Science and Software Engineering
Auburn University
Auburn, Alabama 36849
Email: clinemarka@gmail.com

Matthew Maxwell
Department of Computer Science and Software Engineering
Auburn University
Auburn, Alabama 36849
Email: mrm0053@auburn.edu

Mina Narayanan
Department of Computer Science and Software Engineering
Auburn University
Auburn, Alabama 36849
Email: mjn0010@auburn.edu

***Abstract*—We seek to develop a web crawler that, given a root URL address, will apply Iterative Deepening Search (IDS) to explore the web. The web crawler saves the HTML code of the root web page and its child web pages and performs a character uni-gram feature extraction to develop a feature vector. We tested the web crawler using the URL "http://www.auburn.edu" and a depth limit of 2. Our web crawler worked successfully, outputting thousands of child URLs. In the future, we hope to build upon our improvements to Iterative Deepening Search.**

***Keywords*—Iterative deepening search, web crawler, feature extraction**

I. Introduction

To solve any problem with computational intelligence, a system must be able to find the correct solution. Searching algorithms are the lowest underlying function in most AI programs, so a good understanding of how various ones work and their drawbacks are essential. Iterative Deepening Search (IDS) is a popular strategy that applies a Depth Limited Search (DLS) with increasing depth limits until it reaches some maximum. IDS is capable of finding the optimal solution within time $O(b^d)$.

The benefit of IDS over DLS is that it will find the goal node closest to the root, with the drawback being that it is slower than DLS. In the case of a web crawler, there are no goal nodes, so DLS is the preferred search algorithm.

Many intelligent systems must account for adversaries when performing their tasks. Applying feature extraction on an HTML source produces a feature vector. The developed extractor finds the frequency of each character with relation to the total number of characters in the source.

These tools set the framework for a simple web crawler. Currently, the web crawler is capable of running IDS on a tree of web pages starting from a root URL provided by the user. Successors are hyperlinks found within the page. Each node is run through the feature extractor, which produces a text file detailing the results of the extraction.

II. Methodology

The source code for the web crawler consists of one file written in C++ named webcrawler.cpp and one file written in Java named getWebPage.java.

webcrawler.cpp includes the main driver and two additional functions: find_children() and char_extractor(). find_children() first writes the HTML source of the given URL, accessed via a call getWebPage.java, to an appropriately named text file. Next, if the current URL is not at the depth limit, find_children() parses the HTML source to locate links. These linked URL's, the children of the current URL, are then placed on the stack, and their associated depth is stored on an auxiliary stack. char_extractor() extracts unigrams from the HTML source. The main driver of webcrawler.cpp implements IDS to a user specified depth via a while loop.

In order to obtain HTML source given a web adress, find_children() in webcrawler.cpp passes a URL to getWebPage.java. getWebPage.java then opens a connection to the specified URL and attempts to print the HTML source to the screen, which webcrawler.cpp then pipes to a text file.

Once an HTML source text file is written, a function in webcrawler.cpp, named char_Extractor(), is called. This function reads unigrams from the text files and counts the frequency of each, normalized by the total number of characters in the file. The frequencies are then stored in a separate text file.

A number of improvements were made to the baseline, a naive implementation of iterative deepening search. For one, IDS was not actually used at all. Instead, considering that there are no goal nodes in a web crawler, Depth Limited Search was used, due to its better time complexity. Next, a heuristic was used in which only URLs that begin with the string "http" were expanded. Because the protocol being used to extract HTML source is HTTP, only those web pages which allow access via HTTP are of any use. All others are effectively unscannable lead nodes. An additional constraint was also placed on node expansion in which a parent node cannot have itself as a child. That is, the web crawler ignores links to the current URL. Finally, an improvement to the feature extractor was made in which features were normalized by the total number of unigrams. That is, the frequency of each unigram in a particular HTML source was divided by the total number of unigrams in that source. The benefit of this lies mainly in its utility for future projects. If a machine learning algorithm is applied to these data, web pages with a larger amount of source text

would likely be overweighted if frequencies were allowed to be absolute rather than relative.

III. Experiment

A. Development Phase

The web crawler was first tested using the URL "http://www.auburn.edu" with a depth limit of 2.

B. Improvements

For our three improvements, we 1) implemented Depth Limited Search instead of IDS, 2) only saved URLs that start with HTTP in the stack, and 3) disallowed a node to have itself as a child.

C. Testing

Our first round of testing was to use the same example demonstrated in class, http://www.auburn.edu. After that we looked at various other web pages such as www.spotify.com, www.google.com, www.schema.org, and www.bing.com.

IV. Results

A. Development Phase

While testing the efficiency of our program at a depth limit of 2 using the URL "http://www.auburn.edu, we had 9,879 total links associated with the URL. All .txt files and character frequency vectors were saved correctly into the same directory as the program files.

B. Improvements

For the purposes of Assignment 1, we are not looking to traverse a specific path or achieve a particular goal. The system is reading in URLs to the point we specify as the depth. On the last iteration, the algorithm spans the whole tree that one plans to search, so the previous iterations are redundant. With this realization, we decided to move from IDS to Depth Limited Search.

Many of the hyperlinks did not have the full web address. To solve this issue, we did not load any URLs onto the stack that did not begin with HTTP.

Sometimes, websites may link to themselves. To help alleviate redundant computation, we added a simple check to ensure that pages do not consider themselves as one of their children.

C. Testing

Almost 10,000 nodes were expanded when testing our program with the URL http://www.auburn.edu.

V. Division of Labor

Mina Narayanan wrote the pseudocode for the C++ program, implemented sections of the program including the main driver and parts of the find_children() and char_extractor() methods, and assisted in debugging the program. She wrote the Abstract and parts of the Methodology, Experiment, and Results of the paper. She also implemented the improvement to the program that involved only saving URLs that start with HTTP to the stack.

Matthew Maxwell wrote sections of the C++ program, specifically char_extractor() and parts of find_children(), and debugged the program. He also implemented an improvement to the character extractor in which features, or the frequencies of the unigrams, were normalized by the total number of unigrams.

Mark Cline wrote the Introduction, Testing, and detailed the improvements in the paper. He also wrote the readme.txt file for the webcrawler application.

References

[1] Gerry Dozier

[2] Stuart J. Russel and Peter Norvig, *Artificial Intelligence: Modern Approach.*