Developing an improved focused crawler for the IDEAL project

Client: Mohamed Magdy Gharib Farag

May 8, 2014
CS 4624: Multimedia/Hypertext
Virginia Tech
Blacksburg, VA

Ward Bonnefond, Chris Menzel, Zack Morris
Suhas Patel, Tyler Ritchie, Marcus Tedesco, Franklin Zheng
**Table of Contents**

**Section 1: Executive Summary**
- Executive Summary 1.1 1

**Section 2: User Manual**
- Introduction 2.1 2
- Using the Focused Event Crawler 2.2 3-5
  - Figure 2.1: The Focused Event Crawler web application homepage.
  - Figure 2.2: Full list of results and options after submitting a query.
  - Figure 2.3: Tree view for a submitted query

**Section 3: Developer Manual**
- Introduction 3.1 6-8
- How to start 3.2 9
- Developer Notes 3.3
  - Detailed Overview 3.3.1 10
  - Key Files 3.3.2 11-12
- Front End Overview 3.4 13
- Suggestions for Future Development 3.5 14

**Section 4: Lessons Learned**
- Overview 4.1 15-16

**Section 5: Extras**
- VTURCS Poster 5.1 17
  - Figure 5.1: The poster we presented at VTURCS for Spring 2014

**Section 6: Acknowledgements and References**
- Overview 6.1 18
Executive Summary

The IDEAL (Integrated Digital Event Archive and Library) project currently has a general purpose web crawler to find articles relevant to a set of URLs the user can provide. The resulting articles are return based on frequency analysis of user provided keywords. The goal of our project is to extend the web crawler to return articles related to user provided events and other relevant information. By analyzing an article to identify key event components, such as the date, location, and type of natural disaster, we can construct a tree representation of each webpage. Next, we compute the tree edit distance between that tree, and the event tree constructed from the user’s original input. With this information we can predict webpage relevance with a higher certainty than frequency of keyword analysis provides.

The purpose of this document is to provide a user manual that gives instructions on how to use our web application, and a developer’s manual which provides information on our design decisions, ways to extend the code, and dependencies necessary to host our web crawler implementation.
User Manual

Introduction

The purpose of this user manual is to provide information for anyone interested in using our extended web crawler. Our instructions include figures and details on how to use the web application. While there will be some context on why certain decisions were made among other details, the bulk of the discussion related to interested developers can be found in the developer’s manual.
Using the Focused Event Crawler

Figure 2.1: The Focused Event Crawler web application homepage.

1. Event Type: Choose from a list of all possible natural disasters.
2. Associated Event Name: If the event has a name associated with its occurrence.
3. Location: Any combination of city, state, and country are allowed.
4. Date: Any combination of day, month, and year are allowed.
5. Crawler Type: Choose from the original web crawler or the extended web crawler.
6. Search Page Limit: Enter an integer limit for the total number of pages to search.
7/8. Site URL: Provide base URLs to begin the search from.

A discussion on the scoring algorithm is provided in the Developer’s Manual.
After submitting a query, the results are displayed on the same page below the input section. The output will reflect which URLs were found while searching and accepted after analyzing their content. Below the list of found URLs the total number of accepted out of the total number of found is shown (in this case, 12 out of 20 total webpages were accepted). There is also an option to view the unique tree for each webpage.
Figure 2.3: Tree view for a submitted query.

By clicking ‘View Tree’, the above image is loaded. This provides the user with more specific information on why the page was selected and connects all of the resulting pages together. The tree view also allows the user to verify that the search was performed correctly, and if not it makes all of the relevant data available immediately in order for debugging and other purposes.
Introduction

We started working with the original implementation of the web crawler and made various modifications to it. When talking about the base web crawler, we will refer to it as BaseCrawler for the remainder of this documentation.

BaseCrawler has the following implementation:

Our client provided the BaseCrawler in full functionality. BaseCrawler is a web crawler that crawled the web searching for relevant web pages based on frequency analysis. Setting up the BaseCrawler on a new machine will take a couple hours (see the How to start section for more details). Our new web crawler was built on top of the BaseCrawler files that were supplied to us. Rather than use frequency analysis to score the pages directly, we created a web crawler that took a more focused approach. We will refer to this crawler as FocusedCrawler for the remainder of this documentation. Using parameters passed in by the user, we create an event model that contains the type, location and date of an event.
Figure 3.2: Example of the event model. As you can see Type, Name, Location and Date are all key elements of the tree.

We also pass in seed URLs to begin FocusedCrawler’s search. For every webpage the FocusedCrawler visits, it creates an event model for the page. By comparing the two event models, FocusedCrawler will assign a relevancy score based on the difference between the parameters of each event model. If the score for the current page is higher than a predefined threshold, then we mark it as a relevant webpage and return that to the user at the end of the query.

Figure 3.3: New implementation for the FocusedCrawler.

Overall, we’ve found that our event model produces fewer false positives. Below is a sample of some of the results BaseCrawler found versus FocusedCrawler.
As you can see, even though BaseCrawler accepted more articles out of 100, the types of articles it visited at the end seemed to veer off in terms of relevancy.
How to start

This section will explain how to set up both BaseCrawler and FocusedCrawler on a new machine both to run and to develop and explain our project in more detail.

Prerequisites:

1. Use a machine running a Linux distribution
2. A working Apache server must be installed
3. A recent version of Python must be installed (2.x or 3.x)
4. The following libraries must be installed:
   - PyNER, NLTK, Geonames Cache, zss, pycountry, scipy, numpy, PyYAML, scikit
5. Download the source files from GitHub at:

   https://github.com/wbonnefond/focused-crawler

Installation Steps:

1. Move all the files from focused-crawler/cgi-bin to your machine’s cgi-bin folder.
2. Move all other files to /var/www. If desired you can create a subdirectory to store these files.
3. Edit the javarunner.sh file inside stanford-ner-2014-01-04 directory to change the port the server will be running on (default 8080).
4. Edit EventScorer.py in the get_entities function to match ports with javarunner.sh (default 8080).
5. If Apache server is set up properly, skip this step. Edit index.php: the form HTML element will have an action attribute containing the file path to the FocusedCrawler python file. If this path is wrong, fix the path. In main.js, in the list of ‘include’ statements, there will be an AJAX call. One of the keys of the URL will contain a filepath to the FocusedCrawler Python code. Confirm this is the same filepath.
6. Ensure that cgi-bin is writable and all files within it are writable as well.
7. Start javarunner.sh.
8. Access the webpage by typing localhost/index.php in your webbrowser.
9. Now you can run your query by following the steps found in the user guide.
Developer Notes - Detailed Overview

The BaseCrawler uses frequency analysis to generate a “page relevancy” score. In order for an article to be deemed relevant, the “page relevancy” score must be above the hard coded threshold value. If the page is deemed relevant, the crawler then scans the page for links and calculates a “link relevancy” score based on context, the link itself and the anchor. If this “link relevancy” score is higher than a different predefined threshold, the link is added to the queue.

The FocusedCrawler uses the event model described in the overview (see Figure 3.2 for more details) to calculate the page relevancy score. To create the event model, the PyNEr library pulls date, location and event type from the web page using frequency analysis. It then uses Geonames Cache and pycountry to verify the location information. Adding URLs to the queue is similar to BaseCrawler. The details of the scoring algorithm will be explained in the next section.
Developer Notes - Key Files

FocusedCrawler.py.

FocusedCrawler.py has two key functions:

\[\text{testBaseFC}(\text{seedUrls}, \text{pLimit})\]

\[\text{testEventFC}(\text{seedUrls}, \text{pLimit}, \text{eventTree})\]

Calling TestBaseFC will run the base focused crawler using the seedURLs provided and search through pLimit pages. It will output results similar to figure 3.4.

Calling TestEventFC will run the FocusedCrawler in the same way as TestBaseFC but will also require an instance of the event tree to be passed in as well. See user manual for an example of the input and output of the FocusedCrawler. Note: FocusedCrawler does not perform the actual parsing of the web page text.

As mentioned in the overview, there are some hard coded thresholds that developers can tinker with. To change pageScoreThreshold and urlScoreThreshold threshold values, go to TestBaseFC or TestEventFC and manually change the numbers prior to running the program.

event.py

event.py is simply an object that represents an event. It contains the following attributes:

```
self.event_type=""
self.country=""
self.state=""
self.city=""
self.name=""
self.day=""
self.month=""
self.year=""
self.url"
```

event.py also contains associated function to manage these attributes.
EventScorer.py

EventScorer.py is where the meat of our project was coded. First, EventScorer.py calls get_entities which allows the pyner library to extract the name date and location information. Note, the port needs to be the same port that the javarunner (ner server) is running on.

Then, our modified scoring algorithm builds trees from the user supplied data (see user manual for more details on supplying data) and the current web page and creates a relevance score using the following weighting:

- 25% Date
- 40% Event type
- 35% Location

To calculate the date portion we use the following equation:

date_contrib = \frac{1}{(365 \times \text{diffyears} + 30 \times \text{diffmonth} + \text{diffdays})^{0.2}} \times 0.25

These are experimentally derived numbers that we played with to find the best results.

Event type is weighted by just matching the type. For example, Hurricane Katrina will receive an additional .4 weighting for this category if the url tree also has Hurricane for event type.

Location consists of 35% of the weighting and is calculated in the following way:

- 10% for name if it has a name (i.e. Katrina in Hurricane Katrina)
- 15% for country
- 5% for city
- 5% for state/province

If this score is evaluated to be higher than the threshold score set in FocusedCrawler.py, then the URL is accepted and added to the accepted list. The URLs on the webpage are then evaluated for relevance based on context, anchors and the text of the URL.
Front End Overview

There are many files and interactions used to power the front end of our project, so we will describe the most important files and interactions here. Some basic web development knowledge will be required in order to continue developing the front end.

When the user enters in data on the page inputs and submits the form, the form makes a post to the python FocusedCrawler.py file, utilizing the main.js file, which causes that post to be an AJAX call. When the call completes, the Javascript file updates the page's HTML accordingly with the results. When a button in the results view is clicked, the tree model of the event is displayed. The tree model uses an open source tree javascript library.
Suggestions for Future Development

We found that by only using the event model to crawl the web, we sometimes received more accurate results than BaseCrawler but in other cases we did not. We believe that including keywords pulled from the webpage to create a hybrid base/focused crawler will combine the advantages of both implementations and provide the most accurate list of relevant URLs.
Lessons Learned

While working on this project, the team ran into numerous hiccups in order to be able to complete the project in a timely fashion. Problems arose from basic issues such as scheduling conflicts and time management issues to more advanced issues such as high-level coding and permissions problems. In order to overcome many obstacles, we worked to find solutions that fixed the problems to the best of our ability. However, some problems were unsolvable, yet still provided insight into some of the choices that were made while working on the project.

Initially, we ran into several basic problems that made it difficult to find a starting point for the project. After being given a general overview of our goal – enhancing the original web crawler – we were given source code for the base web crawler with no real starting point. Getting oriented to the project and trying to understand all of the core elements created a very challenging situation. Our client, Mohamed was instrumental in getting us out of that situation, as he was able to explain every facet of the web crawler that we were given, and provided a variety of useful suggestions for where to start working. He was also able to troubleshoot many basic problems we were having by redirecting us to a possible source of information, or giving us steps toward building a workable solution. If we had communicated with him sooner, instead of trying to solve many of these problems ourselves, the initial progress on the project would have gone much smoother.

In addition, early communication issues hampered progress during the early stages of project. Since we were not quite sure what the project required, the team was initially disorganized, and with internal communication virtually nonexistent, it made it difficult to share ideas and insight with one another. Creating a centralized Google Hangout helped solve this problem, as we now had a method of communicating amongst ourselves and with Mohamed that allowed us share information and code extremely quickly. This helped tremendously, as it sped up the debugging process, and allowed us to get directed feedback toward specific problems in a timely manner.

Another problem we faced was that with the sheer amount of code we were initially given, it was extremely easy to add errant code to a file, and then forget the errant code existed until it spawned another problem. We remedied this by using a GitHub repository as our version control. This allowed multiple group members to work on the project simultaneously, and allowed us to rollback changes that were made to the project in case we had code conflicts. This made it so instead of starting over if we ran into an unsolvable issue, we could just roll back to the last time the project performed as intended, and build up from there.

We also ran into platform issues while working on the front-end of the project. Developing on Windows, we hard-coded some values on a source machine that allowed the project to initially work as intended. However, further down the line we learned that these changes were not portable to Linux, and caused problems when run on Linux machines. We then had to redo a significant portion of the front-end to have it work across multiple
platforms, and without any hard-coded values. In addition, we also ran into permission issues between the two different platforms, and thus a significant portion of time was spent rewriting features that were already implemented so they would work on other machines. Through this ordeal, the team understood the value of cross-platform testing, as if we incrementally tested on Linux as well as Windows, we would have caught this problem in the early stages of the project, and would have been able to code the solution correctly in the first place.

Overall, we as a team learned valuable lessons ranging from organizational lessons to proper coding practices. As this project was a learning process, mistakes were bound to happen. We were able to learn from each problem that we faced so that it did not recur, and therefore were able to make progress on finishing the project by a desired deadline.
Extras

Event Aware Focused Crawler
IDEAL Project Case Study

Ward Bonnefond, Chris Menzel, Zack Morris, Suhas Patel, Tyler Ritchie, Mark Tedesco, Franklin Zheng

Client: Mohamed Magdy Gharib Farag

ABSTRACT:
Currently the IDEAL (Integrated Digital Event Archive and Library) project has a general purpose Web crawler to collect articles relevant to seed URLs based on keyword frequency analysis. Our goal is to create a focused crawler tailored to finding events. By analyzing an article to identify key event components, we can construct a tree representation of a webpage. By using the tree edit distance between that tree, and the event tree from seed information, we can predict webpage relevance. Thus, our implementation improves upon general purpose Web crawlers by replacing the “keyword matching” phase with our tree edit distance algorithm. This works since we are able to create a fairly accurate description of an event from a webpage, which can yield more specific information than would keyword analysis alone.

Design Goals:
- More user friendly
- More specific results

Design Decisions:
- Web Interface for maximum portability
- Tree Edit Distance clearly shows why an article is accepted
- Focused on disasters for now

IDEAL Project:
The goal of the IDEAL project is to collect, catalog, preserve, and provide access to services related to digital objects about events, including multimedia — captured from all corners of the Web. Using Web and Tweet collections, the IDEAL project hopes to be able to create unbiased archives for events of interest.

Figure x.x: The poster we presented at VTURCS. We won second place in the Capstone category.
Acknowledgements

We would like to express our gratitude to Mohamed Magdy Gharib Farag (mmagdy@vt.edu) who provided the base files and caught all of us up to speed. His patience, kindness and pleasantness made this project a pleasure to work on. We would also like to thank Dr. Edward Fox (fox@vt.edu) for his guidance throughout the course of the project. We would like to thank the Multimedia, Hypertext, and Information Access students for their insightful suggestions for project improvements. We would also like to thank the judges of the Virginia Tech Undergraduate Research Symposium for acknowledging our project, and for the interest they showed in our work. Finally, we should like to acknowledge NSF IIS - 1319578: Integrated Digital Event Archiving and Library (IDEAL), which gave us the opportunity to work on this project.

References

Python – python.org, and relevant libraries

- PyNER – github.com/dat/pyner
- Natural Language Toolkit (NLTK) – nltk.org
- Geonames Cache – github.com/yaph/geonamescache
- zss – pypi.python.org/pypi/zss
- pycountry – pypi.python.org/pypi/pycountry/
- SciPy – scipy.org
- NumPy – numpy.org
- PyYAML – pyyaml.org
- scikit – scikit-learn.org

Our source code repository can be found at https://github.com/wbonnefond/focused-crawler