Mining and Mapping the Protoweb: A Relational and Graph based analysis

Compiled December 05, 2024.

Shridhara Pavel Rahul Uma ----------

***Abstract*:**. The early 2000s is a foundational era of the internet, widely remembered by static web pages and nascent hyperlink structures. Understanding the structure and behaviour of this era of the internet is vital for tracing the evolution of modern web tech. This project aims to mine and map the Protoweb which is a Open-Source project that contains a catalog of restored sites from the early 2000s by creating a dataset encompassing approximately 3001 pages across 40 domains. Our methodology includes using a custom-built web crawler, metadata such as page title, HTML content, and hyper link structures are extracted and stored in 3 formats which are CSV for raw data, SQLite for relational analysis, and Neo4j for graph based visualization. The relational database model facilitates structured queries and analysis of page interconnections, while the graph database offers useful insight into connectivity patterns, and early concepts such as network traversal. By modeling the Protoweb as a network of nodes and edges, the project reveals vital information regarding the early internet.

I. Introduction

The internet has undergone a massive transformation since its creation, it evolved from a sparse network of static, text based pages to the dynamic and multimedia ecosystem we enjoy today. Despite its simplicity, the early web laid the foundation for modern web technologies and development practices. Exploring the early web provides useful insight into the evolution of digital communication, and the development of information sharing paradigms.

One critical aspect of understanding the web is analyzing its structure and behaviour which is where web crawlers come in. Web crawlers are automated tools designed to traverse and extract data from websites, this task is instrumental when it comes to search search engines, data mining and digital archiving. In the context of the early web, crawlers were important in cataloging content and mapping hyperlinks, providing users with the ability to access information more efficiently.

With this project I aim to design a dataset that maps and mines websites from the early 2000s, capturing approximately 3001 pages or samples and 40 domains. By limiting the scope to a limited number of domains and a maximum crawl depth of 10, I aim to balance the depth of exploration with simplicity and computational feasibility. The dataset construction process relies on a web crawler that maps and mines the early web through the open-source protoweb project that contains a catalog of restored websites from the early 2000s. The crawler will navigate static web pages, collecting essential meta data such as page titles, html, hyperlinks, link captions. The retrieved data will be stored in structured formats, including CSV files for initial simplicity, SQLite for querying and Graph database using neo4j for various analyses.

Beyond just storage, modeling early web data as graph structures with a network of nodes and edges could prove useful in visual and analytical understanding of connectivity patterns. Graph analysis could also prove useful for exploring concepts such as early PageRank and network traversal, which are foundational for search engine algorithms.

II. Methods

A. System Architecture and Design

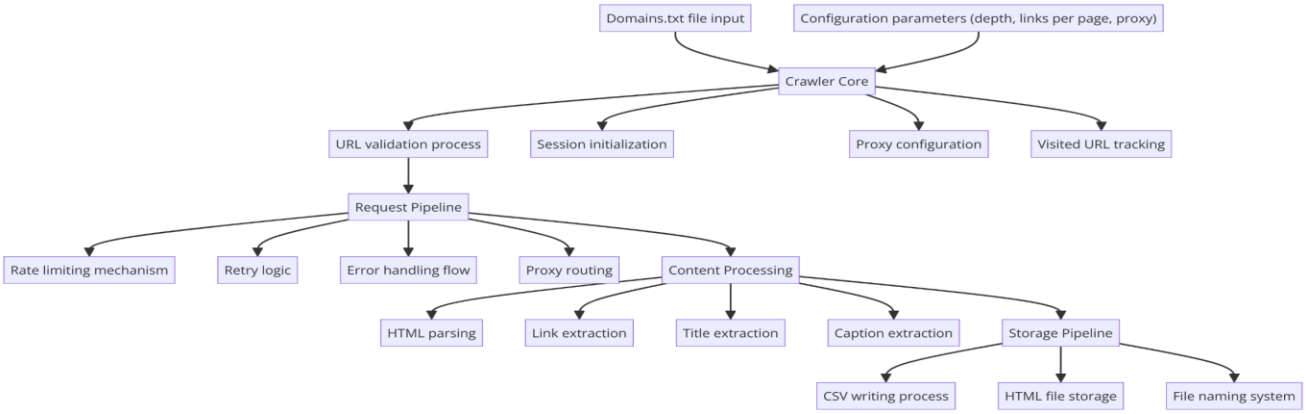
My web crawler implementation utilizes Python libraries such as Beautiful soup to systematically traverse and store web content through a depth-first traversal approach from specified domains. The system architecture consists of multiple interconnected components that work together to crawl and store websites efficiently and safely.

The crawler is built on a class-based system with 3 main components. First the Initialization Component handles setup tasks, managing crawler parameters such as (max_depth), maintaining HTTP connections, organizing output directories, and validating URLs. Second, the Data Collection Pipeline manages URL processing, network requests, HTML content extraction, and link discovery. Third, the Storage system handles data management through CSV writing, HTML content storage, and safe file naming for HTML.

> *The Initialization System*: The initialization system is the crawler's foundation, through the **___init___** method. This function configures crawling parameters such as max_depth, establishes proxy settings, creates necessary directories for HTML files, and prepares the CSV file.
>
> *The Network Management System:* The network management system implements retry logic. It handles various network exceptions, maintains consistent delays or cooldowns between each request, and streams reponses for larger pages. The system's timeout settings are configurable while defaulting to 30 seconds.
>
> *The Content Processing Pipeline:* The content processing pipeline consists of multiple functions for handling different aspects of web content. The link extraction function uses BeautifulSoup for HTML parsing, implements URL normalization, enforces link limits per page, and extracts link metadata. The meta data extraction system consists of two key components, which are page title extraction and link caption extraction, each with fallback logic for data collection.
>
> *The Storage Management System:* The storage management handles both raw HTML content and structured data. For HTML content, it implements a URL based filename sanitization. The structure data storage maintains a schema in CSV format, tracking source and target URL information, domain and depth data, HTTP response metadata, and corresponding filename references.
>
> *Error Handling:* The crawler implements error handling for issues such as connection timeouts, server errors, and proxy failures. It also handles content processing errors such as character encoding issues and file system errors.

The main crawling process begins with the **crawl_url** function. It begins with pre-crawl validation, checking depth limits, and eliminating duplicate URLS. Then it moves through content retrieval, storing raw HTML as files and extracting page metadata. The process continues with link processing, where it extracts and validates links, processes target URLs with a depth-first approach and records the source-target relationships in the CSV file. It implements recursive crawling with a depth-first strategy while maintaining visit tracking.

The final output of the system is a structured dataset containing multiple source-target URL relationships, page metadata including titles and captions, content type information, HTTP data, HTML file system references, and depth data.

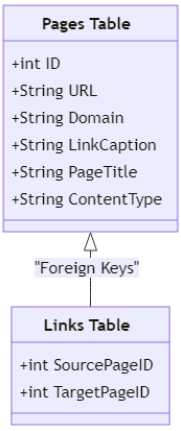


B. Database Preparation

CSV File Structure and Purpose:

The web crawling process generates a CSV file that serves as the raw data repository, it captures the details regarding the URL connections. The CSV file contains twelve distinct columns that document the relationships and characteristics of discovered web links. The key columns include the source and target URLs, link captions, domain information, crawl metadata, and response details. The CSV's primary response extends beyond just data collection, it functions as an intermediary before the data is imported into more complicated database systems such as SQLite and Graph DB.

SQLite Database: Relational Data Management:



Transitioning from the CSV file, the SQLite database offers a more structured and relational approach for data management and analysis. The proposed schema comprises two primary tables, pages and links.

a. Pages: The pages table is the foundational element of the database. Each page is uniquely identified by an auto-incremented integer ID and its URL, domain, link caption, page title, and content type.

b. Links: The companion links table represents the interconnection between each page, establishing a record of hyperlink relationships. By implementing a foreign key constraints to the pages table, the database maintains references while representing link metadata. Each link entry contains source and target page identifiers, link text and caption, and crawl depth.

The database design enables a wide array of possibilities. Researchers can trace complex links between domains, analyze link distribution and connectivity, examine web resource metadata.

Graph Database: Network Structure and Analysis:

The Neo4j implementation transforms the web crawling data into an interconnected network representation. Unlike traditional tabular databases, this graph model captures the relational nature of web pages through a node and edge based structure.

a. URL nodes: Each URL would be a node. Core attributes include, Full URL, Domain, Page title, Content type.

b. Edges or Links: Directed or :LINK_TO relationship between each URL node. Core attributes include, Link text, Link Caption, Crawl Depth.

Researchers can identify most influential web pages, determine hub and authority pages, quantify page importance within the network, reveal interconnected domain groups, highlight community structures, trace interconnection paths, analyze link propagation, discover indirect connections.

C. Challenges and Solutions

These are some challenges I encountered during my implementation, some are solved and unsolved.

a. *Incomplete HTTP response: Handling Partial Data Retrieval:* The crawler consistently encountered IncompleteRead exceptions, where HTTP responses were frequently interrupted or truncated. This issue significantly impacted my data collection process. Initially, the problem prevented data retrieval altogether but I was able to solve this issue by:
   - Enabling streaming in HTTP requests for chunked data processing.
   - Adding robust error handling and retry mechanisms.
   - Used .session() to maintain connection persistence.
   - Introduced timeout and retry logic to manage unstable network state.

   Mitigation Strategy: Despite all this I still faced the issue for complex websites such as Google and Yahoo, limiting the crawl from a list of 300 domains to 40. When the domain breadth became constrained I had to pivot to a depth-first crawling approach. By increasing the crawl depth for accessible domains, the project could still extract meaningful data despite limited domain coverage in the protoweb.

b. *Web Scraping Framework Selection: From Selenium to BeautifulSoup:* Initial attempts using Selenium for web scraping was problematic. The framework's complexity created significant implementation problems, particularly in handling early web content and maintaining parsing capabilities.
   Mitigation Strategy: Transitioned to Beautiful Soup which was lighter and more efficient for the given task.

c. *Proxy and Rate Limiting Complications:* The crawler consistently encountered consistent rate limits or unstable Proxy states.
   - Added a delay between requests.
   - Introduced backoff for failed connections.
   - Configurable Timeout

Mitigation Strategy: More could be done for Mitigation such as dynamic timeout adjustment, proxy rotation, logging of connection attempts and better error handling but I was limited due to my project scope.

d. *Accidental Overwrite:* During the development and testing of the web crawler, critical data management issues occurred which resulted in unintentional data loss. The crawler had successfully collected approximately 6000 samples with corresponding HTML content across 70 domains. However, during script iteration and testing, I ran the script without modifying the output file name which resulted in complete data overwrite.
Consequences: Irretrievable loss of initial collection and significant time investment rendered useless.
Mitigation Strategy: Partial recovery was only possibly through a separately maintained CSV without HTML content and much lesser number of samples.

e. *Domain Boundary Oversight and Scope Drift:* Another significant oversight occurred during the web crawl. Initially, the crawler included domain boundary logic to ensure it crawled with a list of predefined domains listed in a domains.txt file. However, due to other issues, the task of removing this logic was forgotten and the mistake went unnoticed. This problem was only discovered after a night of scraping and dataset preparation. When the dataset was re-scraped without boundaries to compensate, it still failed to reach the desired sample size due to limited time constraints.

f. *Time Constraints:* Despite working on this project for weeks I wasn't able to finish the analysis I wanted to do with the SQLite database nor was I able to reach the Sample Size I desired. The dataset could also be greatly enriched and improved with some minor bug fixes and some more time to build a dataset without domain boundaries.

III. Results and Data Analysis

In this section, we explore the results of the mine and map of the Protoweb. I particularly focus on two main aspects: the data collected and the graph-based analysis of the early internet's structure. The dataset formed through our web crawler and its subsequent analysis provide important information regarding the early web, particularly regarding how websites were linked and the roles they played in forming the networks of the internet in the early 2000s.

There are a total of two datasets, First a dataset with domain boundaries with a sample of 3900 pages spanning 40-50 different domains, the second dataset with a sample of approximately 500 pages spanning 25 domains. I was unable to build the second dataset in time due to an error during the crawling process (refer to challenges and solutions). The crawling process collected a variety of metadata, including page titles, content types, hyperlink relationships, and additional crawl-related metadata. The raw data was captured in CSV format, which provided a simple tabular representation of the relationships between different pages. Then I transformed and transferred this data into both a relational SQLite database and a graph-based Neo4j database for further analysis.

a. Graph Analysis and Visualizations:

One of the most valuable outcomes of this project was the conversion of raw web page and hyperlink data into graph representations in neo4j that illustrated the early web's topology. With Neo4j, we could conduct several types of graph-based analysis. Each URL became a node with several attributes and properties while the hyperlinks between pages were represented as directed edges.

i. I conducted a rudimentary PageRank analysis to identify the most influential nodes within the network. These pages could be seen as central hubs in the early web but due to the domain boundary issue I could only perform this on the smaller dataset which failed to yield insightful results.

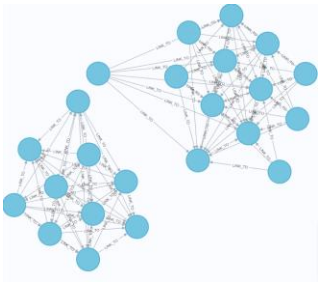
ii. The graph structure allows us to study network traversal and the depth of connectivity between pages. I discovered some domains that had higher levels of connectivity compared to others, which could be an indicator of how search engines could rank pages based on the link structure but again due to domain boundaries this analysis was not reliable.

To further illustrate my findings, I created a few visualizations using Neo4j's built-in tools such as bloom and cypher queries. These visualizations offer a picture of how the early internet was interconnected and how certain websites and pages played significant roles in navigating the web.

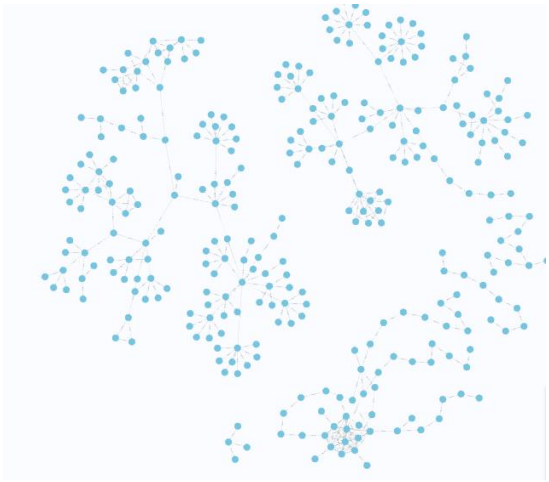
1. Visualization 1: Network Topology of the Protoweb: This graph shows the entire network of 3900 pages for the bigger dataset with domain boundaries. Each node represents a URL, and the directed edges represent the hyperlinks between each web page.

Visualization 2: The second visualization breaks the graph down to clusters that represent specific domains. The clustering of web pages based on domains, with certain domains having more tightly knit pages while others having barely any connectivity or no connectivity.

Visualization 3: The third visualization displays two tightly knit domains or a set of pages.



Visualization 4: The fourth visualization displays the entire network topology of the smaller dataset with no domain boundaries.



b. Relational Analysis: Exploring the SQLite Database:

The dataset contains a total of 3011 pages and 3,782 links with an average crawl depth of 7.24 and a max crawl depth of 10. It's a well balanced dataset in terms of depth and breadth. Although a richer dataset across a wider set of domains could be built with more time.

1. Top 10 Most Common Domains: The analysis of domain frequency highlights the most prominent domains within the dataset.

Domain	Page Count
cd.textfiles.com	698
simpsonsarchive.com	583
nethack.org	298
www.cornica.org	280
theoldnet.com	233
system7today.com	180
falconfly.3dfx.pl	107
nofi.mariteaux.somnolescent.net	88
retrosite.org	76
www.warpstream.net	62

2. Content Type Distribution: This is the content type reveals the variety of sources crawled.

| Content Type | Count |

text/html	932
text/html; charset=utf-8	449
text/html; charset=UTF-8	385
[empty] (unclassified content)	286
text/html; charset=iso-8859-1	233
application/zip	167
application/x-msdownload	126

The majority of pages are HTML-based, with various encodings but formats like application/zip suggest the presence of downloadable software or archives in the protoweb.

V. Future Directions

The project opens up numerous possibilities for further research:

a. *Extended Domain and Temporal Scope*: Expanding the list of domains and content from other periods would provide a better view of the web's evolution. This would involve tackling the issues related to incomplete responses and scaling the crawler for a broader coverage.
b. *Dynamic Content Crawling:* Enhancing the crawler to handle dynamic content with frameworks like selenium could enrich the dataset with more interactive elements of early websites that static crawlers like mine may miss.
c. *Integration with Machine Learning:* Applying machine learning techniques to predict patterns in early web connectivity or reconstruct missing data could provide new insights that might go unnoticed with traditional analysis.
d. *Publication:* Publishing the dataset as an open-source resource for researchers and hobbyists could cause a more comprehensive exploration of the early web. However, before doing so, I would like to improve and enrich both the crawler and the dataset.

VI. Conclusion

Through this project, I have successfully mapped and mined a subset of the protoweb or the early web, offering a relational and graph-based analysis of its structure. By using a systematic web crawler and multiple data storage methodologies, including CSV, SQLite, and Neo4j, it provides a comprehensive dataset and analysis. Despite challenges such as incomplete HTTP responses and limited domain breadth, I was able to successfully extract meaningful data and analysis. This project contributes to the broader goal of capturing the essence of the early web, preserving its unique characteristics and offering useful insights into its evolution. While my crawler achieved a reasonable amount of success with the data, there remains significant potential for improvement and promising directions for both the data and the mining.