CS 703

Project Proposal

Paul Luh

Problem Statement

This project aims to implement a system that can synthesize `XPath` query for `DOM` data extraction from `HTML` pages. Given a `HTML` page and a set of annotated `DOM` nodes on the page, the system will automatically learn `XPath` query that can extract `DOM` nodes from different `HTML` pages but consistent with the patterns found in the user-provided examples.

From the perspective of information extraction, the set of user-provided `DOM` nodes can be considered as *mentions* for a single *entity type*. For example, if the *entity type* is `President of the United States`, the set of user-provided examples can be tree nodes of {`Bill Clinton`, `Barack Obama`, `Donald Trump`}.

As the project progresses, I will answer the following questions throughout my investigation.

(a) How efficient is the synthesis algorithm proposed in the reference papers [1]? What are the bottlenecks?

(b) Are there any limitations in terms of expressiveness when using `XPath` query to extract information for real-world web pages? Do we need to make any simplicity assumption about the page structure so that `XPath` can perform competitively?

(c) How many examples are needed to have acceptable performances from the algorithms in the reference paper? Can we simply choose the examples arbitrarily or are there any patterns that we need to follow?

Motivation

As more data are available on the Internet, collecting data from semi-structured web pages becomes an integral process when obtaining training data for machine learning applications, sources for knowledge base construction, and pieces of evidence for business analysis. However, although many web pages have similar structure and the information of interest typically shares similar patterns in those web pages, the development of web scrapers is still considered a labor-intensive activity since one need to (1) design the schema in which the extractor will populate (2) observe the patterns in the web page structure that can be utilized when writing web scrapers (3) develop the web scrapers

and handle corner cases (4) update the web scrapper when the website provider changes the layout. Therefore, there are always great interests from the research community in developing a mechanism to automate the process of web data extraction [2]. This process is typically coined in the data mining community as *wrapper induction*.

Key Components

Document Parser

The document parser will parse the `HTML` pages into in-memory representation on which the synthesis engine can execute. The parser will build on top of out-of-box `HTML/XML` parser from existing `Python` libraries. However, I expect minor postprocessing steps on the parse tree so the synthesis algorithm can operate properly.

Annotator

The annotator will map input annotations to the corresponding parts in the in-memory representation.

Synthesis Engine

A `Python` implementation of the graph-based synthesis algorithm from [1].

Output Finalizer

Since the output of the synthesis engine will be a graph program, the output finalizer will translate the original program into `XPath`.

Experiments

I plan to conduct the following two lines of experiments.

(a) Data extraction from Wikipedia infobox. Since infoboxes on Wikipedia pages have very similar structures and the presentation is relatively compact, this experiment is designed to demonstrate the engine's abilities to handle homogenous format from data sources.

(b) Attribute extract from product pages from `Amazon`. This experiment is designed to stress test the engine if the webpages have more content and may contain noisy information.

I will measure the performance following the metrics typically used in information retrieval application namely `Accuracy, Recall, F1-Score`. The access of ground-truth is not clear at this point; I will figure this out when the project progresses. In the worst case, I will annotate them manually.

Milestones

Deliverable 1 Parser that can parse the document into in-memory data structures. I need to fully understand the algorithm in the paper first so I know what representations are suitable for the synthesis algorithm.

Deliverable 2 Synthesis engine that can operate on the in-memory data structures given user-provided annotations.

Deliverable 3 Experiments for testing the synthesis engine.

Appendix: Progress Report

Implementation Plan

1. (completed) Understand the semantic of the augmented treveral graph in the paper [1].
2. Understand the semantic of the remaing data structure in the paper [1].
3. (completed) Design a data structure that this algorithm can be performed. (It turned out to be an augmented graph data structure)
4. (completed) Crawled Wikipedia pages from the internet. (I crawled the infobox section of the U.S. President's Wikipedia page)
5. Crawled product pages from Amazon.com.
6. (work in progress) Implement a compiler that can transform the raw representation into the data structure designed above.
7. On top of the data structure, implement the algorithms investigated in bullet point 1.

Experimentation Plan

1. Try extracting the information about the political party of each president using the program learned by the algorithm. Will test the performance with different number of examples.
2. Try extracting the price information from several product pages from Amazon.

References

[1] Tobias Anton. Xpath-wrapper induction by generating tree traversal patterns. In *LWA*, 2005.

[2] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[3] Vu Le and Sumit Gulwani. Flashextract: a framework for data extraction by examples. In *PLDI*, 2014.

[4] Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. Automated migration of hierarchical data to relational tables using programming-by-example. *Proceedings of the VLDB Endowment*, 11(5):580–593, 2018.