Proposal for CS4246R Project: Markov Decision Process for Focused Web Crawling

U096883L Shawn Tan

April 13, 2012

1 Introduction & Motivation

The amount of content on the World Wide Web has been growing exponentially. Systems have been built to cope with this large amount of data by either categorising them, or indexing them to make search possible.

With the introduction of more user-generated content in recent years, efforts have been made not only to classify the web but also toward understanding the data available. Data-mining usually involves crawling websites and analysing the data using machine learning techniques.

A naive approach to crawling usually involves maintaining a FIFO queue in which links seen are stored, and visited either sequentially, or as mentioned previously, requested when a data stream is unused. All of the links seen on the resulting page or pages would then be added to the FIFO queue, and the process is repeated until all pages in the site have been visited, or a sufficient crawl depth has been reached. Sifting through *all* of the links incurs a huge I/O cost when only a small percentage of the entire site are the target pages. This is when focused crawling techniques are used.

Focused crawling makes use of information given on the visited page to try to determine which links are relevant for further exploration and which to discard. This reduces the cost of fetching irrelevant pages (e.g. fetching a site's about page when crawling for product prices).

Crawling is also largely a task that requires some programming knowledge to perform. In order to crawl the site in an efficient manner, a programmer has to specify which links to follow, or add to the FIFO queue, at any given page. The job of reinforcement-learning based crawlers is to figure out during the crawl which links are relevant to follow, cutting out the need for a programmer to explicitly state the links to be followed.

In this report, we experiment with using a particular MDP-based reinforcement learning algorithm, the Least-Squares Policy Iteration We achieve results slightly better than the naive version.

2 Proposed Approach & Implementation

I intend to investigate how modeling not only the uncertainty of the transitions, but also the uncertainty of the current state the crawler or bot is in using POMDPs would affect its performance, and its overall efficiency.

My UROP topic was based on Web Information Extraction, and when POMDPs were introduced during the course of CS4246, I thought it would be interesting to apply it to focused web crawling. There has been work done using reinforcement learning to control a web crawler [?, ?, ?], but to my knowledge, none for POMDPs. I would like to investigate how accounting for state by giving it a probability distribution would improve the performance of the crawler.

The project is likely to be a web crawler implemented in Python. One of my learning goals in implementing the project, is to understand what are the various ways POMDPs are solved or approximated. In doing so, I aim to use the method most suitable for this particular problem. The process of framing the as a POMDP also requires mapping of state-actions or state-action-states to rewards. This is another detail with regards to using POMDPs that I would like to look further into.

To evaluate the system, I intend to employ the methods used in Rennie, 1999 [?]. Given a site, I will identify similar pages of the same type of data, using this as the target set of pages for each crawler to find. Two different control crawlers will be implemented:

Optimal This crawler will be scripted by hand to navigate directly from the start point to the various target pages to be downloaded.

Breadth-First This crawler will use a breadth-first search to explore the site, retrieving the target pages only when it finds them

The efficiency of the crawler will be measured by the number of target pages it finds over time. Another similar metric could also be the ratio of the retrieved target pages to the number of total pages retrieved. This is a reliable metric as the closer the ratio tends towards one, the lesser the I/O cost required to get the target pages.

So talk about:

- what web crawling is. why we need reinforcement learning for the task + Tedious to write scraper. Need to do a new one for every single new site we're scraping. + Scraping typically requires programming knowledge + What if we could have the crawler learn how to crawl the site rather than have the developer write a new site-specific crawler every time.

- Related work. + Reinforcement-based crawlers:how they work + Use naive bayes classifiers to classify link text into different bins of estimated Q-values. Then uses the bins to calculate expected q-value to decide what links to follow at every stage. + How our approach is different + Uses LSPI. + Different way of using textual content information for reinforcement-learning

- Results + Uber shitty.

- Discussion + Textual information super important + Link information also quite important + Using visual data is difficult because difficult to use least-squares regression to come up with a model for specific matching of numbers. + Improvements - Use textual information (Cite regina barzilay) in a smarter way, but yet, still in a reinforcement learning framework. - Better framework useable by computer n00bs. click click some shit, then let the crawler run and figure the shit out. - Possible to generalise a script that can be run standalone independent of the framework?

-Conclusions - Results like fuck. - Why the results like fuck? + Because the difficult to use textual content in linear least-squares regression model. - Learnt a lot of shit.