ENGL 398 Proposal:
Software Research Plan for an Artificially Intelligent Spider

Gregory Ziegan, Computer Science
April 17, 2014

Submitted to—
Claire McBroom

Abstract

The goal of this study is to propel the author to begin the research and development of an artificially intelligent web spider. Outlined below is the detailed overview, showcase for its need to be implemented, and a coordinated plan aimed at resolving the nebulous creation process. After time spent successfully coding this application, the author will have something of value to present to his company. Besides the inherent value from presenting this asset to his company, the process of implementing this scale project will result in a host of new technical experiences.

Project Description

Overview

This proposal acts as the basis of a research plan which is meant to orient the author to a software engineering topic known as web spidering. Execution of this research plan will increase the author's acclimation to the realm of intensive, independent software development. The team effort will only remain in terms of accountability and performance feedback.

The scope of this project involves executing a plan to educate the researcher on artificial intelligence software.

This plan is limited by the uncontrollability of the domain it spiders. Bad HTML could cause many failures. As a result, the research plan does not detail the progress of every possible step in its development, as large complications are expected given the unreliability of the spider's working set.

Included in this proposal is a literature review, which describes, in detail, some of the tools and practices the spider will implement. Also included is the research plan, and a schedule of work. The proposal concludes with a discussion about the qualifications of this researcher, followed by a budget for the researcher's company to fund.

Literature Review

Introduction

This review of literature will focus on the practice of implementing artificially intelligent learning software, focusing on its application towards spidering websites. Research performed for this review will be conducted with a knowledge of existing techniques, classic AI algorithms, and private spidering frameworks. An implementation of this research would lighten the workload of my fellow co-workers considerably and develop a kind of spider not publicly known to be implemented anywhere today.

Justification of Utilizing a Functional Programming Language

"The nearest thing Common Lisp has to a motto is the koan-like description, 'the programmable programming language,'" recounts Peter Seibel in his book, Practical Common Lisp. The beauty of a functional language like Common Lisp is its propensity towards meta-programming. Lisp macros are unlike any other macro in the programming world, allowing for a compile-time generation of a domain-specific language. [3] Since the domain I will be working with will be exclusively HTML and Javascript source code,

customizing a Lisp spider to natively understand these languages will be essential. Building a library in a dynamic, imperative language might be faster in terms of development time, but the prototyping time in Lisp will all but make up for time lost due to unfamiliarity with the language's syntax. [6]

Ultimately, choosing a language for a programming assignment is arbitrary. Arbitrary, but there are typically reasons one language would be better to use than another. In general, the topic is debatable but a programmer has to choose based on his research. I chose a functional language not only for its ability to express complex algorithms with as little boilerplate as possible but also for its clarity when I am unsure of the entire extent of the programming problem. Expressivity and power are paramount in the stage of prototyping artificially intelligent software. [4]

Spidering

A spider is a combination of a web crawler and parser. A web crawler follows hyperlinks, constructing a memory of pages visited and doing something with that knowledge. A guided crawler is typically used when only relevant pages are desired. An example of a guided web crawler would be one that is programmed to retrieve all the product pages on a retail website. The crawler would not care about any page that is not an individual product. A parser simply extracts desired information from a page of code. Since these are web pages, they are rendered with a combination of HTML and Javascript. Using the properties of these languages, a parser would need to be able to find specific information on the page encoded in these languages. For instance, a parser could be programmed to extract the address of a business from a web page and save the information to a database. Combining these two pieces of software results in a spider. [1]

When spidering sites for specific information, each spider needs to be handcrafted. A programmer needs to put time into detailing exactly what the spider will need to expect on each page, including both the hyperlinks it will follow and the information it will parse. While it seems hopeless to make such a process automated due to the massive variety of ways a web page could be written, there are common patterns. Wherever there exists repetition and some amount of consistency, it is worthwhile to look into whether a program can be written to recognize the patterns.

Techniques to Use

I will use three important tools to implement an intelligent spider: a large knowledge base, a variation of the Naive-Bayes classifier, and the CLOS object paradigm specific to Common Lisp.

A knowledge base refers to what the spider already knows. This will consist of information pertaining to each specific problem it faces. For instance, if a parser if looking for addresses, it can refer to a table full of all the existing U.S. street names, street prefixes and suffixes, and cities. As more addresses are encountered and approved, they can be added to the knowledge base. The knowledge base will be known as dynamic and monitored by the programmer. [5]

The Naive-Bayes classifier is a commonly used algorithm that classifies problems based on the likelihood of such a problem falling into a specific pre-determined category. Imagine three colors that balls coming out of a factor could be. If a programmer designed something to sort the balls by weight and the color was somehow correlated to weight, after initially programming what probabilities different colors have corresponding to specific weights, the algorithm would group future balls based on the probabilities. It is a learning algorithm since the probabilities will update after classifying each ball based on a Bayesian network. [2][5]

CLOS is Common Lisp's object oriented programming system. It contains generic functions and methods that make use of both the functional style and other more commonly used language' object oriented systems. This will be useful in classifying new spiders and new functions for these spiders by linking them with a base spider. The base spider will define the common techniques for navigating websites and parsing pages. [6]

Conclusion

Utilizing all of the aforementioned techniques will guarantee the production of a semi-intelligent spider. The heuristics designed due to previous exposure to spidering will be what separates this artificially intelligent spiders from previous attempts. Past attempts are not often recorded as they could easily be trade secrets and very valuable to an individual company. Most previous research is outside the realm of academics as there has yet to be significant reason for universities to profit from further research in this area.

Research Plan

This project will take approximately four months to complete, between May 5th, 2014 and August 25th, 2014. This time scale is very rough and based on the assumption that the researcher's company will not allow him to work solely on research and development instead of other client-facing products. The end date of the project is listed as August 25th because fall semester back at school begins the next week for the researcher.

May 2014:

- Research Common Lisp
- Implement successful “dumb” spider

June 2014:

- Implement Naive Bayes Classifier
- Develop training sets
- Look for feedback from spider/modify code

July 2014:

- Test parsing element on real web searches
- Begin development for crawling portion based on new classifier

August 2014:

- Test on large scale sites
- Q/A data and hand off to other developers

Qualifications

Given the researcher’s year of experience of industry and near-senior status upon the completion of this current semester, he would be an ideal candidate for the research and programming of this challenging project. Since his company deals so heavily with this software niche of spidering websites, he is at a precipice of innovation considering his access to and understanding of private frameworks. A solid foundation in computer science concepts also elevates his qualifications based on the quality of school and side projects he has coded in his last few years at Case.

Budget

Included below is the tentative budget to execute the research plan (at the company’s discretion):

Pay per hour……. $25-30
Number of hours/week……20
Number of weeks working…..15

Total…….$7500-9000

No other overhead besides time spent working would be required since all known techniques and algorithms are available for free online. A computer to develop this spider has already been provided for by the researcher’s company.

References

[1] Tanimoto, S. (1987). *The elements of artificial intelligence*. Retrieved from http://www.osti.gov/scitech/servlets/purl/6214268

[2] "Naive Bayes Classifier". Retrieved January, 2013 Available: http://www.statsoft.com/textbook/naive-bayes-classifier

[3] Seibel, Peter , Practical Common Lisp. Apress ed , Vol . .2005.

[4] Norvig, Peter , Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp. 1st ed , Vol . .1991.

[5] Russell, Stuart , Artificial Intelligence: A Modern Approach. Pearson 3rd ed , Vol . .2010.

[6] Barski, Conrad , Land Of Lisp. No Starch Press ed , Vol . .2010.