# Using Reinforcement Learning to Spider the Web Efficiently[*]

**Jason Rennie**[†‡]
jrennie@justresearch.com

[†]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Andrew Kachites McCallum**[‡†]
mccallum@justresearch.com

[‡]Just Research
4616 Henry Street
Pittsburgh, PA 15213

## Abstract

Consider the task of exploring the Web in order to find pages of a particular kind or on a particular topic. This task arises in the construction of search engines and Web knowledge bases. This paper argues that the creation of efficient web spiders is best framed and solved by reinforcement learning, a branch of machine learning that concerns itself with optimal sequential decision making. One strength of reinforcement learning is that it provides a formalism for measuring the utility of actions that give benefit only in the future. We present an algorithm for learning a value function that maps hyperlinks to future discounted reward using a naive Bayes text classifier. Experiments on two real-world spidering tasks show a three-fold improvement in spidering efficiency over traditional breadth-first search, and up to a two-fold improvement over reinforcement learning with immediate reward only.

## 1 INTRODUCTION

Spiders are agents that explore the hyperlink graph of the Web, often for the purpose of finding documents with which to populate a search engine. As information extraction techniques improve, spiders are also becoming vital elements in the automated creation of Web-based knowledge bases.

Extensive spidering is the key to obtaining high coverage by the major Web search engines, such as AltaVista and HotBot. Since the goal of these general-purpose search engines is to provide search capabilities over the Web as a whole, they aim to find as many distinct web pages as possible. Such a goal lends itself to strategies like breadth-first search. If, on the other hand, the task is to find pages of a particular kind or on a particular topic (as is the case for the increasingly popular class of domain-specific search engines), then an intelligent spider should try to avoid hyperlinks that lead to off-topic areas, and concentrate on links that lead to documents of interest. Similarly, if the goal is to locate documents for use in populating a topic-specific knowledge base, much effort would be wasted by following every hyperlink that is found.

This paper describes our research in efficient, topic-directed web spidering. We argue that the creation of efficient web spiders is best framed and solved by reinforcement learning, a branch of machine learning that concerns itself with optimal sequential decision making. One strength of reinforcement learning is that it provides a formalism for measuring the utility of actions that give no immediate benefit, but do give benefit in the future. Reinforcement learning agents represent this delayed benefit by learning a mapping from each available action to a scalar value indicating the sum of future discounted rewards expected from executing that action. The "discount" makes later rewards less valuable than sooner rewards, thus encouraging efficiency.

In our current reinforcement learning spider, we learn a mapping from the text in the neighborhood of a hyperlink to the expected (discounted) number of relevant pages that can be found as a result of following that hyperlink. The mapping from the text to a scalar is performed by casting regression as classification [Torgo and Gama, 1997]. Specifically, we discretize the scalar values into a finite number of bins, and use naive Bayes

[*]Published in proceedings of ICML-99 Workshop, Machine Learning in Text Data Analysis

to classify the text into a corresponding finite number of classes; the value assigned to a particular hyperlink is a weighted average of the values of the top-ranked bins.

Our research in efficient spidering is part of a larger project that has created Cora, a domain-specific search engine that indexes computer science research papers [McCallum *et al.*, 1999]. We have spidered computer science departments and labs, so far finding over 50,000 research papers in postscript format. Each of these papers is converted to plain text, then run through a specially-trained hidden Markov model to automatically find the title, authors, abstract, references, etc. Forward and backward references between papers are resolved. Finally, all this information is made available in a searchable, public web interface at *www.cora.justresearch.com*.

In Cora efficient spidering is a significant concern. The majority of the pages in many computer science department web sites do not contain links to research papers, but instead are about courses, homework, schedules and admissions information. Avoiding whole branches and neighborhoods of departmental web graphs can significantly improve efficiency and increase the number of research papers found given a finite amount of crawling time.

Our work is also driven by the WebKB project [Craven *et al.*, 1998]. Here the focus is on automatically populating a knowledge base with information that is available on the World Wide Web. The system is given an ontology consisting of object classes and relations of interest (*e.g.* company, location, CEO, is-located-in, is-officer-of), and also a training set consisting of labeled instances of these classes and relations (*e.g.* web pages labeled by class, and web page pairs labeled by relation). The system then aims to extract automatically new instances of these classes and relations from the Web. Before the system can extract such instances, it must locate the web documents containing the appropriate information. This is where spidering becomes important. By spidering efficiently we can increase the size of the attainable knowledge base in the face of ubiquitous limitations of time, computation and network-bandwidth. In this paper we spider to locate the Web page listing the chief executive officers of a company.

We report on experiments performed on data sets taken from both the Cora and WebKB domains. Our experiments show that reinforcement learning is a highly effective framework for the spidering problem. In both cases, reinforcement learning outperforms traditional spidering with breadth-first search by a factor of three or more. For both data sets, we find that explicitly modeling future reward provides significant benefit. The nature of the benefit depends somewhat on the context. In the case of Cora, where there are many reward-providing documents in each spidering run, modeling future reward provides benefit in the first half of the run only—before many hyperlinks leading to immediate reward have been discovered. In our WebKB task, however, each spidering run contains only a single reward document (as in common reinforcement-learning "goals of achievement"), and thus many decisions must be made based on future reward. As a result, our approach that explicitly models and predicts future reward can find the desired documents more efficiently. In our WebKB experiments, modeling future reward increases efficiency by a factor of two or more.

## 2 REINFORCEMENT LEARNING

The term "reinforcement learning" refers to a framework for learning optimal decision making from rewards or punishment [Kaelbling *et al.*, 1996]. It differs from supervised learning in that the learner is never told the correct action for a particular state, but is simply told how good or bad the selected action was, expressed in the form of a scalar "reward."

A task is defined by a set of states, $s \in \mathcal{S}$, a set of actions, $a \in \mathcal{A}$, a state-action transition function, $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and a reward function, $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$. At each time step, the learner (also called the *agent*) selects an action, and then as a result is given a reward and its new state. The goal of reinforcement learning is to learn a *policy*, a mapping from states to actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that maximizes the sum of its reward over time. We use the infinite-horizon discounted model where reward over time is a geometrically discounted sum in which the discount, $0 \leq \gamma < 1$, devalues rewards received in the future. Accordingly, when following policy $\pi$, we can define the *value* of each state to be:

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r_t, \qquad (1)$$

where $r_t$ is the reward received $t$ time steps after starting in state $s$ and following policy $\pi$. The optimal policy, written $\pi^{\star}$, is the one that maximizes the value, $V^{\pi}(s)$, for all states $s$.

In order to learn the optimal policy, we learn its value function, $V^\star$, and its more specific correlate, called $Q$. Let $Q^\star(s,a)$ be the value of selecting action $a$ from state $s$, and thereafter following the optimal policy. This is expressed as:

$$Q^\star(s,a) = R(s,a) + \gamma V^\star(T(s,a)). \qquad (2)$$

We can now define the optimal policy in terms of $Q$ by selecting from each state the action with the highest expected future reward: $\pi^\star(s) = \arg\max_a Q^\star(s,a)$. The seminal work by Bellman [1957] shows that the optimal policy can be found straightforwardly by dynamic programming.

## 2.1 SPIDERING AS REINFORCEMENT LEARNING

As an aid to understanding how reinforcement learning relates to spidering, consider the common reinforcement learning task of a mouse exploring a maze to find several pieces of cheese. The agent receives immediate reward for finding each piece of cheese, and has actions for moving among the grid squares of the maze. The state is both the position of the mouse and the locations of the cheese pieces remaining to be consumed (since the cheese can only be consumed and provide reward once). Note that the agent only receives immediate reward for finding a maze square containing cheese, but that in order to act optimally it must choose actions considering future rewards as well.

In the spidering task, the on-topic documents are immediate rewards, like the pieces of cheese. An action is following a particular hyperlink. The state is the set of on-topic documents remaining to be found, and the set of hyperlinks that have been discovered. The state does not include the current "position" (last page visited) of the agent since a crawler can jump to any known URL next. The number of available actions is large and dynamic, in that it depends on which documents the spider has visited so far. It is as if the mouse can leap to any square, as long as it has already visited a bordering square. The key feature of topic-specific spidering that makes reinforcement learning the proper framework is that the environment presents situations with delayed reward.

## 2.2 PRACTICAL APPROXIMATIONS

The problem now is how to apply reinforcement learning to spidering in such a way that it can be practically solved. Unfortunately, the state space is huge and difficult to generalize. It encapsulates not only the on-topic documents remaining to be found, but also the set of hyperlinks that are available as actions. It is not immediately apparent how one would construct a metric to compare different states. The action space can also be large: the number of distinct hyperlinks within the Web that we are considering.

We need to make some simplifying assumptions in order to make the problem tractable and to aid generalization. Note, however, that by first defining the exact solution in terms of the optimal policy, and making our assumptions explicit, we will better understand what inaccuracies we have introduced, and how to select areas of future work that will improve performance further. We make simplifying assumptions that: (1) disregard state and (2) capture the relevant distinctions between the actions using only the words in the "neighborhood"[1] of the corresponding hyperlink. By disregarding state, we generalize across all states and can determine a single $Q$-value for each action; by assigning a set of words to each hyperlink we can generalize across different hyperlinks by comparing the text around them.

Thus our $Q$ function becomes a mapping from a "bag-of-words" to a scalar (discounted sum of future reward). Learning to perform efficient spidering then involves only two remaining sub-problems: (1) obtaining training data consisting of bag-of-words/$Q$-value pairs, and (2) learning a mapping using the training data. These are each described in the following two subsections.

## 2.3 OBTAINING TRAINING DATA AND CALCULATING $Q$-VALUES

There are several choices for how to gather training data. Although the agent could learn from experience on-line, we currently train the agent off-line, using collections of already-found documents and hyperlinks. For these training sets, we know the state transition functions, $T$, and the reward functions, $R$. With this knowledge, we can obtain bag-of-words/$Q$-value pairs and use these to estimate the $Q$-function for an unexplored portion of the Web.

Recall that the $Q$-value of a hyperlink is the discounted sum of rewards received by following the optimal policy after traversing the hyperlink in question.

[1]Text that is related to the hyperlink in question, for example, the unordered list of words in the anchor, headers, and page title associated with the hyperlink.

When calculating $Q$-values for tasks with a single goal and single reward, dynamic programming produces for each hyperlink a value equal to the reward, discounted by its distance to the goal. This calculation can be done directly without running full dynamic programming.

When there are many rewards, however, the process is more complicated. Rather than trying to fully represent the exponentially-sized state space and applying dynamic programming, we again aim to calculate the optimal sequence directly and set the Q-value to the discounted sum of this sequence. We calculate the optimal sequence directly by maintaining a "fringe" of hyperlinks available from all the pages in the sequence so far, and greedily traversing to the reward closest to the fringe. Initially, the fringe only contains the hyperlink whose $Q$-value we are currently calculating. Choosing among the fringe represents the fact that a spider is not required to select a hyperlink from among a last page visited, but can choose among any of the hyperlinks of which it is aware (*i.e.* the fringe). Note furthermore that the fringe in this calculation is based only on the pages reachable from the initial hyperlink; in practice at testing time, the agent will have already explored other hyperlinks, and will be able to "jump" to any of these.

To see why a greedy search is reasonable, consider assigning $Q$-values to two hyperlinks, $A$ and $B$. $A$ yields a reward of 1; $B$ yields a reward of 0 and reveals a gold mine of hyperlinks, each of which gives a reward of 1. Imagine a spider that had only these two hyperlinks as possible actions. If it follows $B$ first, its sequence of rewards would be $\{0, 1, 1, \ldots, 1\}$. Otherwise the rewards would be $\{1, 0, 1, \ldots, 1\}$. Since we value efficiency, $A$ is the better choice. In fact, unlike many RL domains where moving towards immediate reward can take the agent further from a gold mine, here all alternative actions are available without futher cost, and it is always best to first follow a hyperlink that yields immediate reward. Thus, we choose a $\gamma$ that mandates this policy (given a perfect $Q$-function predictor), $\gamma = 0.5$. This causes nearer rewards[2] to be more valuable and makes greedy search an appropriate way to determine an optimal action sequence.

## 2.4 MAPPING TEXT TO $Q$-VALUES

Given that we have calculated $Q$-values for hyperlinks in our training data, next we must learn a generalized value function that maps hyperlinks to scalar values. The mapping should be efficient, and should generalize to future, unseen hyperlinks.

We represent the value function using a collection of naive Bayes text classifiers, performing the mapping by casting this regression problem as classification [Torgo and Gama, 1997]. We discretize the discounted sum of future reward values of our training data into bins, place the text in the neighborhood of the hyperlinks into the bin corresponding to their $Q$-values, and use the hyperlinks' neighborhood text as training documents for a naive Bayes text classifier. For each new hyperlink that we find, naive Bayes yields a probabilistic class membership for each bin based on the neighborhood text for that hyperlink. The Q-value associated with each bin is set to the average of the Q-values associated with the training hyperlinks assigned to that bin. Then the value of a new hyperlink is estimated by taking a weighted average of each bin's $Q$-value, using the probabilistic class memberships as weights. This gives us a method for using naive Bayes as a function approximator.

We next introduce the naive Bayes text classifier. Approaching the task of text classification from a Bayesian learning framework, we assume that text data is generated by a parametric model, and use training data to calculate maximum a posteriori estimates of the model parameters. Equipped with these estimates, we classify new test documents using Bayes' rule to turn the generative model around and calculate the posterior probability that each class would have generated the test document in question. Classification is then the simple matter of selecting the most probable class given the document's words.

The classifier parameterizes each class separately with a document frequency and with word frequencies. Each class, $c_j$, has a document frequency relative to all other classes, written $\mathrm{P}(c_j)$. Each class is modeled by a multinomial over words. That is, for every word, $w_t$, in the vocabulary, $V$, $\mathrm{P}(w_t|c_j)$ indicates the frequency that the classifier expects word $w_t$ to occur in documents in class $c_j$.

We represent a document, $d_i$, as an unordered collection of its words. To classify a new document with this model we make the naive Bayes assumption: the words in the document occur independently of each other given the class of the document, (and furthermore independently of position). Using this assumption, classification becomes straightforward. We calculate the probability of each class given the evidence of

[2]Those obtainable through a shorter sequence of actions.

the document, $\mathrm{P}(c_j|d_i)$, and select the class for which this expression is the maximum. We write $w_{d_{ik}}$ for the $k$th word in document $d_i$. We expand $\mathrm{P}(c_j|d_i)$ with an application of Bayes' rule, and then make use of the word independence assumption:

$$\begin{aligned} \mathrm{P}(c_j|d_i) &\propto \mathrm{P}(c_j)\mathrm{P}(d_i|c_j) \\ &\approx \mathrm{P}(c_j)\prod_{k=1}^{|d_i|}\mathrm{P}(w_{d_{ik}}|c_j). \end{aligned} \tag{3}$$

Learning these parameters ($\mathrm{P}(c_j)$ and $\mathrm{P}(w_t|c_j)$) for classification is accomplished using a set of labeled training documents, $\mathcal{D}$. To estimate the word probability parameters, $\mathrm{P}(w_t|c_j)$, we count over all word occurrences for class $c_j$ the frequency that $w_t$ occurs in documents from that class. We supplement this with Laplace 'smoothing' that primes each estimate with a count of one to avoid probabilities of zero. Define $N(w_t, d_i)$ to be the count of the number of times word $w_t$ occurs in document $d_i$, and define $\mathrm{P}(c_j|d_i) \in \{0, 1\}$, as given by the document's class label. Then, the estimate of the probability of word $w_t$ in class $c_j$ is:

$$\mathrm{P}(w_t|c_j) = \frac{1 + \sum_{d_i \in \mathcal{D}} N(w_t, d_i)\mathrm{P}(c_j|d_i)}{|V| + \sum_{s=1}^{|V|}\sum_{d_i \in \mathcal{D}} N(w_s, d_i)\mathrm{P}(c_j|d_i)}. \tag{4}$$

The class frequency parameters are set in the same way, where $|\mathcal{C}|$ indicates the number of classes:

$$\mathrm{P}(c_j) = \frac{1 + \sum_{d_i \in \mathcal{D}} \mathrm{P}(c_j|d_i)}{|\mathcal{C}| + |\mathcal{D}|}. \tag{5}$$

Empirically, when given a large number of training documents, naive Bayes does a good job of classifying text documents [Lewis, 1998]. More complete presentations of naive Bayes for text classification are provided by Mitchell [1997] and McCallum and Nigam [1998].

## 3 EXPERIMENTAL SETUP

In August 1998 we completely mapped the documents and hyperlinks of the web sites of computer science departments at Brown University, Cornell University, University of Pittsburgh and University of Texas. This comprises our Cora data set; it includes 53,012 documents and 592,216 hyperlinks. The 2,263 target pages (for which a reward of 1 is given) are computer science research papers. They are identified with very high precision by a hand-coded algorithm that checks for abstract and reference sections.

Our WebKB data set includes 6,643 web pages, the complete web sites of 26 companies. These companies were selected randomly from a pool of over 1000 companies in a WebKB knowledge base. Here, the target page is one that includes information about officers and directors of the company. One such page was located by hand for each company, thus giving us a total of 26 target pages.

An advantage of choosing these two data sets is that they have extremely different properties. For Cora, the task is to find a large number of target pages in a single spidering run, and many rewards may be found connected to the same web page. Thus accurately identifying immediate rewards is very important. Each WebKB graph, on the other hand, contains only a single target page. In order to locate this single reward, the spider must traverse a path of hyperlinks. Along this path, the spider must compare the $Q$-values of many hyperlinks; thus, distinguishing future reward is very important.

We use two different types of *neighborhood text* for our regression by classification task. The first we call *full-page* neighborhood text and associate with each hyperlink two bags-of-words with separate vocabularies: (1) the full text of the page on which the hyperlink is located, and (2) the anchor text of the hyperlink and portions of the URL. The second we call *related* neighborhood text and associate with each hyperlink four bags-of-words with separate vocabularies: (1) headers and title words, (2) the anchor text of the hyperlink, (3) directories and filenames in the URL of the hyperlink, and (4) a small set of words immediately before and after the hyperlink. One should note that our choices of neighborhood text were hand selected and were not learned in any way. In ongoing work, we are experimenting with other types of neighborhood text to examine their their affect on spidering performance.

For each experiment that we describe, we perform a series of runs, one for each department or company in the data set. The spider is trained using data from all other departments or companies and is tested on the Web of the remaining one. For comparison, we also present the results of two spiders that use no form of learning. The Breadth-First spider follows hyperlinks in the order it finds them using a FIFO action queue. The Optimal spider has full knowledge of the spidering graph and always follows the path to the nearest reward.

The RL Immediate spider uses binary classification for regression. One bin includes all hyperlinks with a $Q$-value of 1 (those hyperlinks that point directly at target pages). The second bin includes all other hyperlinks. If our $Q$-value approximator were perfect, this spider would perform breadth-first search until it found a hyperlink that points directly to a target page. Note, however, that probabilistic classification provides intermediate values for text that could be in either class. This is a weaker form of regression for representing future reward than a multi-binned classifier; the estimated value of a future reward hyperlink will only be high if it shares features with immediate-reward hyperlinks.

The RL Future spider uses future reward as described in section 2.3, and performs regression by classifying into three or more bins—one or more for varying amounts of future reward. We experimented with between three and five bins. For example, our four-bin classifier maps immediate-reward ($Q = 1$) hyperlinks into one bin, hyperlinks with $1 > Q \geq \gamma$ into a second bin, hyperlinks with $\gamma > Q \geq \gamma^2$ into a third bin, and all remaining hyperlinks ($\gamma^2 > Q$) into a fourth bin.

The main difference between RL Immediate and RL Future is that RL Future has one or more bins for representing future reward hyperlinks. For example, in our three-bin RL Future spider, the second bin represents hyperlinks that point to a web page with immediate-reward hyperlinks. On the other hand, RL Immediate distinguishes only immediate-reward hyperlinks from all other hyperlinks.

## 4 RESULTS

Here we present the results of experiments performed using a spider that is adequately able to parse and navigate through HTML frames. Previous versions of our spider did not have this capability and hence the results presented here may differ somewhat from results presented in previous work.

Figure 1 (Top) shows results from our different spiders on the Cora data set. Notice that at all times during their progress, the reinforcement learning spiders have found more research papers than Breadth-first. One measure of performance is the number of hyperlinks followed before 75% of the research papers are found. Reinforcement learning performs significantly more efficiently, requiring exploration of only 14% of the hyperlinks; in comparison Breadth-first requires 43%. This represents a factor of three increase in spidering efficiency.

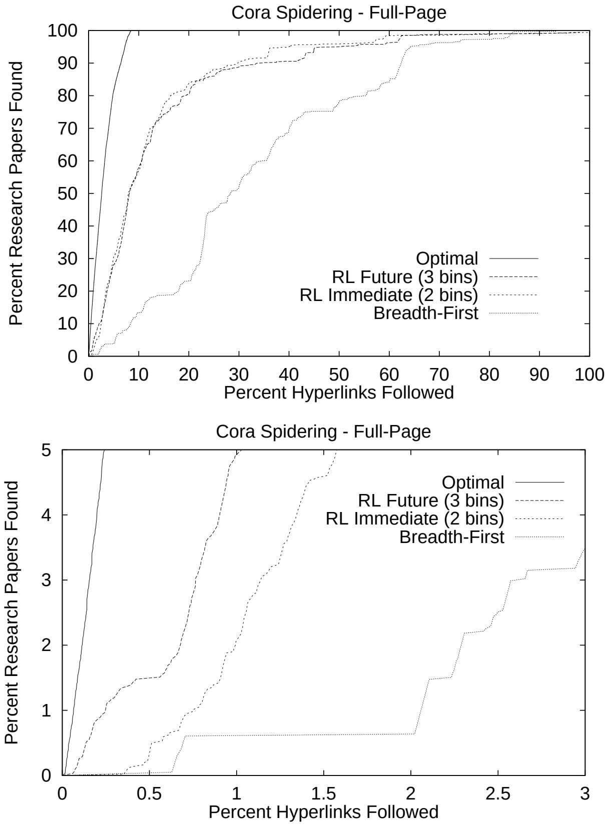
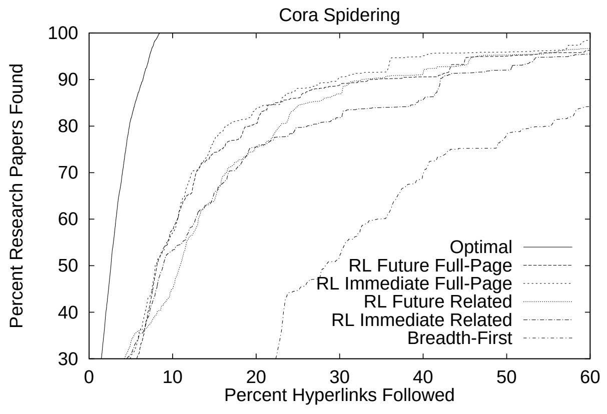


Figure 1: **(Top)** A comparison of spiders on the Cora dataset using *full-page* neighborhood text with results averaged over the four departments. The reinforcement learning spiders retrieve three-quarters of the research papers in one-third the page expansions that Breadth-First requires. Note that RL Future performs better than RL Immediate in the earliest stages of spidering. **(Bottom)** A close-up near the origin of (Top).

Note also that the RL Future spider performs better than the RL Immediate spider in the beginning, when future reward must be used to correctly select among alternative branches, none of which give immediate reward. RL Future locates more than 10% of the target documents more quickly. This early period is important because many applications of spidering require that only a sample of all target pages are retrieved. Figure 1 (Bottom) shows a closeup of the early part of the run. On average RL Future takes significantly less time than RL Immediate to find the first 28 (5%) of the papers—efficiently avoiding an average of 350 extra page downloads that are performed by RL Immediate.

In Figure 1, after the first 30% of the papers are found,



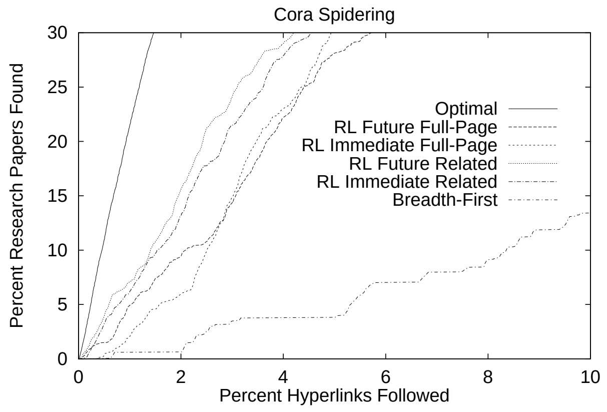


Figure 2: A close-up of spidering performance with results on *full-page* and *related* neighborhood text. Note the skewed scales. **(Bottom)** displays results through the locating of the first 30% of research papers. **(Top)** displays results from the later stages of spidering. Spiders using *related* neighborhood text retrieve the first 30% of research papers more quickly than the spiders that use the *full-page* neighborhood text.

the RL Immediate spider performs somewhat better than the RL Future spider. This is because the system has uncovered many links that will give immediate reward if followed, and the RL Immediate spider recognizes them more accurately. On average, RL Immediate is able to identify immediate reward links with 95% recall, while three-bin RL Future only achieves 88% recall.

A similar phenomenon is seen when comparing spidering performance using different notions of neighborhood text. Figure 2 (Bottom) shows that, during the early stages of spidering, a spider using *related* neighborhood text performs better than one using *full-page* text. However, as spidering continues (Top), the roles are reversed and the *full-page* spiders perform better. As was mentioned before, the RL Immediate *full-page* spider achieves 95% recall. The *related* version of this spider only achieves 71% recall on classifying research paper hyperlinks. On the other hand, the *related* spider achieves greater overall accuracy due to a high recall score for the classification of other hyperlinks. These results lead us to believe that high recall in classifying research paper hyperlinks leads to better performance in the middle and later stages of spidering. By this time, the spider has found many hyperlinks that point to research papers. Since it can better identify these hyperlinks, it more readily follows them. During the early stages of spidering, identifying future reward hyperlinks is more important; since RL Future and spiders that use *related* text do better at this task, they are quickly able to find paths toward research paper hyperlinks at the beginning of the spidering task. Since applications that can benefit from efficient spidering require the retrieval of only a fraction of all on-topic documents, we venture to say that *related* is a better choice of neighborhood text than *full-page.* Attempting to *learn* an optimal, domain-specific definition of "neighborhood text" is one potential area of future research.

While we have examined the results of spidering runs, we have yet to explore the workings of the internal engine, the classifier-regressor. Such analysis gives us some insight into spidering performance and leads us toward future interesting experiments. The overall classification accuracy of the underlying classifier in the RL Immediate (2 bin) classifier is generally better than that of the RL Future (3 bin) and *much* better than the classification accuracy of the RL Future (4 bin) spider; the 2-bin RL Immediate averages 85% accuracy while the 3-bin and 4-bin average 77% and 59%, respectively. While accuracy does not directly tie to spidering performance (3-bin performs about as well as 2-bin), it does hint toward the added complexity in a classification problem with more bins; the benefits of using more bins must outweight the increased difficulty in order that spidering performance benefits. Interestingly enough, if one considers the 2-bin classification task using the 3- or 4-bin RL Future spider (mapping the 3 or 4 labels given by the RL Future spiders into the 2 labels given by the RL Immediate spider), one achieves *increased* classification accuracy. Using the 3-bin RL Future classifier for the 2-bin task, we achieve 92% accuracy. Using the 4-bin classifier, we achieve 95% accuracy. This is to be compared against the rate of 85% achieved by the 2-bin classifier. We conjecture that this increased accuracy in the 2-bin task may increase spidering performance.

Table 1: Spidering performance on WebKB data.

| SPIDER | PCT LINKS FOLLOWED |
|---|---|
| Optimal | 3% |
| RL Future (4 bins) | 13% |
| RL Future (3 bins) | 22% |
| RL Future (5 bins) | 27% |
| RL Immediate | 27% |
| Breadth-First | 38% |

We now turn to the WebKB data set, where reward is much more sparse, and we expect future reward to be more important. Table 1 shows the results of different spiders on the WebKB data. As hoped, a spider that uses distinctions among future reward achieves the best results by far. On average, the four-bin RL Future spider is able to locate the officer page after traversing only 13% of the hyperlinks within the company. This is twice as efficient as RL Immediate, which follows an average of 27% of the hyperlinks before locating the target page. In further contrast, Breadth-First follows an average of 38% of the hyperlinks before finding each officers page.

Note also that the four-bin classifier performs better than the three-bin. There is a tradeoff, however, between the flexibility of the classifier-regressor and classification accuracy. Experiments with a five-bin classifier result in worse performance—roughly equivalent to the RL Immediate spider, following an average of 27% of available hyperlinks before locating the target page. Better features and other methods for improving classifier accuracy (such as shrinkage [McCallum *et al.*, 1998]) should allow the more sensitive multi-bin classifier to work better.

In order to provide a window into successful four-bin classification for the value function, Table 2 shows the ten most predictive words per class (ranked by weighted log-odds ratio) for the four-bin RL Future spider[3] The column at the top ("immediate") refers to the class containing immediate rewards; the column at the bottom ("other") refers to the class representing the least amount of reward. The parenthesized label indicates in which part of the hyperlink neighborhood each word was found[4].

[3]"aerial," "beckman," "burr," and "brown" are company names in our data.

Table 2: Most predictive words for RL Future (4 bins)

$Q = 1$
**IMMEDIATE**

| | |
|---|---|
| board (url) | corpinfo (url) |
| information (url) | executive (link) |
| directors (link) | team (link) |
| beckman (url) | board (link) |
| anrpt (url) | management (link) |

$1 < Q \leq 0.5$
**TWO-STEP**

| | |
|---|---|
| aboutbb (url) | corp (url) |
| bb (link) | inc (head) |
| annual (url) | annual (head) |
| about (link) | report (head) |
| nbsp (near) | inside (url) |

$0.5 < Q \leq 0.25$
**THREE-STEP**

| | |
|---|---|
| applications (url) | siteindex (url) |
| aerial (head) | employment (url) |
| burr (head) | position (head) |
| brown (head) | open (head) |
| sales (url) | administration (head) |

$0.25 < Q \leq 0$
**OTHER**

| | |
|---|---|
| clp (url) | and (near) |
| elk (url) | region (head) |
| doc (url) | to (near) |
| elk (head) | nbsp (near) |
| datasheet (url) | sl (url) |

Note that predictive words in the "immediate" class are directly relevant to a page that lists officers and directors of a company. We can induce from the "two-step" class that annual reports and "about" pages commonly include hyperlinks to a listing of company officers. The "three-step" class includes words concerning employment and product releases. It is common to include a link to the annual report on such pages.

To this point, we have described experiments that use a classifier-regressor built by partitioning hyperlink $Q$-values. The semantics of a hyperlink's neighborhood text does not necessarily correspond to the hyperlink's $Q$-value. Thus, it may be advantageous to build a classifier-regressor using classes that are based on textual semantics. We could achieve such a semantically-partitioned set of classes via statistical clustering such

[4]"Url" refers to text in the URL. "Head" indicates header and title words. "Link" refers to anchor text. "Near" indicates text found shortly before or after the hyperlink.

as the methods described in [Hofmann and Puzicha, 1998]. We believe that such a modification will vastly improve our classification accuracy and, as a result, improve spidering performance.

## 5 RELATED WORK

Several other studies have also researched spidering, but without a framework defining optimal behavior. ARACHNID [Menczer, 1997] is a system that uses a collection of agents for finding information on the Web. Each agent competes for limited computational resources, procreating and mutating proportionally to its success in finding relevant documents. Information gathering experiments are demonstrated on the Encyclopedia Britannica Propaidia tree and synthetic data. By contrast, our spider has roots in optimal decision theory, and searches unstructured pages from the real Web.

Cho *et al.* [1998] introduce a heuristic metric, PageRank, for valuing a web page based on its linkage properties. They show that PageRank is an effective spidering metric for locating pages with high PageRank counts or back link counts. However, these metrics perform poorly when the task is to locate pages that are relevant to a particular topic or query. Our research focuses on exactly that aspect: creating a framework to locate web documents that are related to a particular topic.

Additionally, there are systems that use reinforcement learning for non-spidering Web tasks. WebWatcher [Joachims *et al.*, 1997] is a browsing assistant that helps a user find information by recommending which hyperlinks to choose. It thus restricts its action space to only hyperlinks from the current page. WebWatcher uses a combination of supervised and reinforcement learning to learn the value of each word on a hyperlink. Our work is not user-centric and strives to find a method for learning an optimal decision policy for locating relevant documents when hyperlink selection is unlimited.

Laser [Boyan *et al.*, 1996] is a search engine that automatically optimizes a number of parameters to achieve improved retrieval performance. The CMU CS Web is used as the test bed and evaluation is based on the user's selection of links presented by Laser. The work finds that incorporating HTML markup into the TFIDF weighting scheme improves retrieval performance. Utilizing such markup may also be effective for further improving spidering performance.

## 6 CONCLUSIONS

Our results provide strong evidence that reinforcement learning is an excellent framework within which to perform Web spidering. Experimental results on two data sets show a three-fold improvement in spidering efficiency over traditional breadth-first search. We find that modeling future reward is particularly important when reward is sparse. This occurs in two important problem classes: (1) when the spider is trying to locate one or a proportionately small number of target pages in a large web graph (*e.g.* WebKB), or (2), when the spider is in the first half of a run in which there are many target pages (*e.g.* Cora). In these cases, explicitly modeling future reward provides a two-fold increase in efficiency.

Several areas of future work have already been mentioned. In particular, we are currently improving the classifier accuracy so that RL-Future out-performs the immediate spider even in cases with dense immediate rewards. Additionally, we plan to investigate other value function criteria that relax some of our current assumptions.

**Acknowledgements**

We thank Kamal Nigam, Kristie Seymore and Mark Craven for their insightful discussions and their feedback on earlier drafts.

## References

[Bellman, 1957] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Boyan *et al.*, 1996] Justin Boyan, Dayne Freitag, and Thorsten Joachims. A machine learning architecture for optimizing web search engines. In *AAAI workshop on Internet-Based Information Systems*, 1996.

[Cho *et al.*, 1998] Junhoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Computer Networks and ISDN Systems (WWW7)*, volume 30, 1998.

[Craven *et al.*, 1998] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Sean Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[Hofmann and Puzicha, 1998] Thomas Hofmann and Jan Puzicha. Statistical models for co-occurrence data. Technical Report AI Memo 1625, Artificial Intelligence Laboratory, MIT, February 1998.

[Joachims *et al.*, 1997] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the World Wide Web. In *Proceedings of IJCAI-97*, 1997.

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, pages 237–285, May 1996.

[Lewis, 1998] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *ECML-98*, 1998.

[McCallum and Nigam, 1998] Andrew McCallum and Kamal Nigam. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998. http://www.cs.cmu.edu/∼mccallum.

[McCallum *et al.*, 1998] Andrew McCallum, Ronald Rosenfeld, Tom Mitchell, and Andrew Ng. Improving text clasification by shrinkage in a hierarchy of classes. In *ICML-98*, pages 359–367, 1998.

[McCallum *et al.*, 1999] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Building domain-specific search engines with machine learning techniques. In *AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, 1999. http://www.cs.cmu.edu/∼mccallum.

[Menczer, 1997] Filippo Menczer. ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery. In *ICML '97*, 1997.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[Torgo and Gama, 1997] Luis Torgo and Joao Gama. Regression using classification algorithms. *Intelligent Data Analysis*, 1(4), 1997.