A Topic-Specific Web Robot Model Based on Restless Bandits

Constructing and maintaining topic-specific Web indexes is modeled by a restless-bandits generalization and resolved by a reinforcement-learning algorithm.

Tadhg O'Meara and Ahmed Patel
University College Dublin

Web search engine design is primarily concerned with two distinct processes: ranking and indexing.[1] *Ranking* returns a list of the most relevant documents in response to a given query. Efficient ranking requires *indexing*, in which search engines construct and maintain a database, or index, of available documents.

Document acquisition can follow either a push or pull model. In the push model, publishers submit documents to a search engine for indexing. In the pull model, search engines acquire documents. Web robots—Web crawlers or spiders—acquire documents from Web servers by following hyperlinks. Robots require little or no cooperation from document publishers, and give search engines control over what is indexed. Today, most robots attempt to build an index of all documents on the Web, or of a representative sample. In the future, however, the use of topic-specific Web robots, which automatically build and maintain indexes of topically related Web pages, will increase significantly.

In this article, we outline the potential role of topic-specific robots in distributed search engine design, and we model the complex problem of automatically constructing and maintaining topic-specific Web indexes. Experimental results establish the viability of a topic-specific Web robot design based on the restless bandit model. The results indicate that our proposed algorithm is a good foundation on which to build a complete solution.

A Distributed Search Architecture

Search engine design that can scale with Web growth is a long-standing research goal. Today's predominant engines (such as AltaVista, Fast, Google, and Inktomi) employ a centralized search architecture. Each provides a ranking service for all queries in the search services market. The ranking, indexing, and database components of these engines can be distributed across many computers. Efficient distribution is achieved by enabling the ranking and indexing processes to access and con-

1089-7801/01/$10.00 ©2001 IEEE

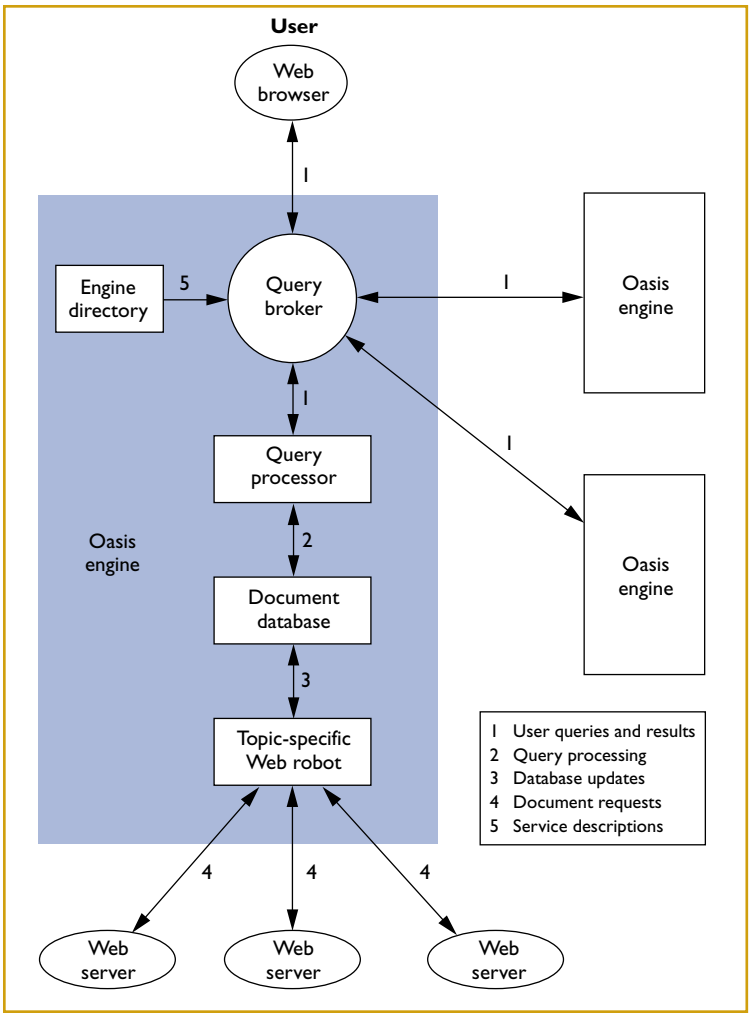

Figure 1. The Oasis architecture. The architecture consists of a distributed system of search engines, each specializing in a selected topic. Engines cooperate by propagating queries not relevant to their selected topics and compete for queries through their selection of topics.

trol the complete document database.[2] Consequently, although these systems scale to large numbers of computers and can process and index numerous queries and documents, such scaling requires centralized control of the underlying database.

As the Web continues to grow, the network and hardware resources required to build and maintain a database of the entire Web are becoming too complex and expensive for any single organization. In a survey of the most popular search services, Lawrence and Giles estimated that no single engine indexed more than 16 percent of the publicly indexable Web.[1] Combined, the 11 largest engines indexed only 42 percent.

Alleviating this bottleneck requires an architecture that enables a scalable distribution of the ranking and indexing loads in the presence of multiple ownership and distributed control of the underlying databases. An example architecture is the Open Architecture Server for Information Search, a distributed search engine research project. The goal of the Oasis project was to enable a scalable, low-entry-cost market of competitive search services.[3] To achieve this, the Oasis consortium designed an architecture and protocols for a distributed system of independently owned and controlled topic-specific search engines.

Oasis: Topic-Specific Search

Figure 1 shows the basic Oasis architecture. Each engine ranks and indexes only queries and documents relevant to a selected topic. The ranking and indexing functions are performed by the query processor and Web robot components, respectively.

Each engine can directly access and control only its own document database. Engines, however, can propagate queries to remote engines using their query brokers, allowing indirect access to remote databases through the associated engines' ranking services. The engine directory component provides advance knowledge of a remote engine's services. Engines cooperate by propagating queries that are not relevant to their own database. Engines compete for queries through their selection of topics and the quality of their databases and ranking mechanisms.

To enable a scalable distribution of the ranking and indexing processes, we chose a topic-specific distribution of the system database. In this system, each query need be propagated only to a few of the most relevant engines. This contrasts with a non-topic-specific system where each engine is equally likely to index documents relevant to any given query. Consequently, in systems like Desire,[4] in which the database is distributed by geographic or network domain, each query must be propagated to and processed by all engines.

Open Problems

A system based on topic-specific engines does have its problems. Efficient query routing, for example, requires accurate advance knowledge about an engine's topic orientation. Another problem is automatic service management—that is, how can each engine automatically select its own topic specialization for the benefit of all?

It is also much harder to build and maintain Web databases by topic than by geographic or network domain. One approach pools acquired document descriptions using standard Web robots and selects documents from this central pool in a topic-specific manner. This approach can be

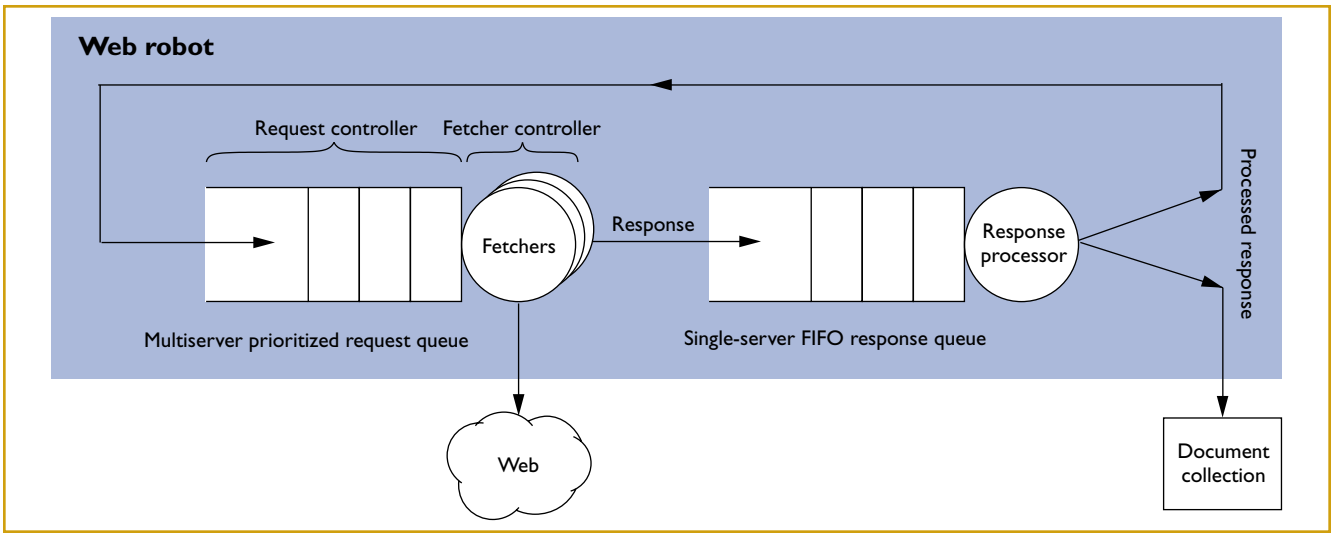

Figure 2. Basic Web robot model. The problems of deciding which documents to request and how many concurrent requests to make are resolved by the request and fetcher controllers, respectively.

implemented by systems like Harvest.[5] Such an approach, however, makes the resultant databases' quality a function of collective network and hardware resources. For truly independent ownership and control, each engine must construct and maintain its own database. In the Oasis project, we designed a heuristic-based, topic-specific Web robot algorithm for this purpose.[6] In this article, we describe a more formal approach in hopes of obtaining closer to optimal performance.

Topic-Specific Web Robots

A topic-specific Web robot aims to schedule document requests to maximize the quality of its topic-specific document database as quickly as possible. For convenience, we refer to a topic-specific document database as a *collection*.

A collection's quality derives from the relevance of its document entries and the number of out-of-date entries. Since there is an optimal request schedule for any given relevance scoring function, we do not consider the accuracy of the function here. Naturally, however, any topic-specific Web robot is only as good as its scoring function; fortunately, the information retrieval field has extensively researched this problem. The function used in this article is based on the standard vector space model and has been shown to provide good results.[6]

The topic-specific Web robot problem can be decomposed into two separate decisions:

- what documents to request, and
- how many concurrent requests to make.

Concurrent requests are necessary to fully utilize system throughput by hiding network latency.

Model Components

Figure 2 illustrates a basic Web robot model. The main components are the request controller and the fetcher controller. The *request controller* decides what documents to request by scheduling requests using a prioritized queue. The *fetcher controller* decides how many concurrent requests to make by adjusting the number of document fetchers serving the request queue. When a fetcher finishes with a request, it places the Web server's response in a buffer called the response queue.

The response processor services the response queue and outputs the results to both the request controller and the document collection. The request controller results include relevance scores and links extracted from retrieved documents. The document collection results include instructions to add, modify, or delete document entries. The request controller schedules requests to include the most relevant documents in the collection and to discover modifications to existing collection documents as soon as possible.

Two sources impose constraints on this schedule:

- Due to the graph structure of documents on the Web, a document cannot be requested unless a document containing a link to it has previously been retrieved.
- A robot sending a large number of requests to the same Web server in rapid succession, called *rapid fire requests*, can cause degradation in Web server performance. To avoid RFRs, a Web

robot must observe a given minimum time between requests to the same server.

We can model the fetcher controller as a single-server queuing problem. The controller attempts to minimize both the response processor's idle time and the response queue's size. To do this, the controller adjusts the response arrival rate by deciding, on either a response arrival or departure event, whether to service a new request. This decision is based on the number of requests currently in service and the size of the response queue. We are investigating the use of average reward reinforcement-learning algorithms to solve this problem.

Performance Evaluation

Because we want to maximize the collection's quality as soon as possible, we can calculate the performance of any request schedule as the integral of the resultant collection quality over the schedule's duration. This calculation lets us focus on collection quality measures at any time.

To specify a performance measurement for a request schedule, we derive measurements similar to those used in information retrieval. We assume that all retrieved documents are added to the collection. The only limit on collection size is imposed by the combination of limited system throughput and the requirement that entries be kept up-to-date.

We define two main facets of collection quality:

- *recall* refers to the number and relevance of documents in the collection with respect to the documents on the Web;
- *freshness* refers to the number of out-of-date document entries in the collection with respect to the collection's size.

Collection recall and freshness are defined analogously to the standard measurements of recall and precision employed in information retrieval. We define the recall and freshness of collection C at time t as

$$R(C_t) = \frac{\sum_{i \in (C_t \cap V_t)} r(i)}{\sum_{i \in V_t} r(i)}, F(C_t) = \frac{\sum_{i \in (C_t \cap V_t)} r(i)}{\sum_{i \in C_t} r(i)} \quad (1)$$

respectively, where $r(i) \in \Re^+$ is the relevance score of document i, C_t is the set of document entries in the collection, and V_t is the set of documents on the Web at time t. Document modifications are viewed as simultaneous additions and deletions to V_t. We define collection freshness as a function of document relevance because we assume that the importance of maintaining an up-to-date entry is proportional to its relevance to the collection.

Given that recall and freshness measure two different aspects of collection quality, it is not clear how to measure collection quality in terms of a single number. We propose to measure a collection's quality as the minimum of its recall and freshness $R(C_t)$ and $F(C_t)$:

$$M(C_t) = \min [R(C_t), F(C_t)] , \quad (2)$$

where

$$P^{\mu} = \int_0^T M\left(C_t^{\mu}\right) dt \quad (3)$$

is the performance of the request schedule μ of duration T, and C_t^{μ} is the collection state resulting from μ at instantaneous time t.

Directly optimizing this measurement in the online case is problematic because the set of documents existing on the Web at any given time is unknown. Instead, we use P^{μ} to evaluate the performance of schedules generated for a known subgraph and try to maximize the expected discounted reward function.

Dynamic Programming Algorithms

Reinforcement learning refers to a broad collection of techniques including learning automata and simulation-based dynamic programming (DP) algorithms.[7] We are interested here in the latter.

The principal elements of a DP problem are

- a dynamic system whose state transition depends on a control, and
- a cost or reward that accumulates additively over time.[8]

When at state x, the control u must be chosen from a given set $U(x)$. At state x, the choice of a control u specifies the state transition probability $p_{xy}(u)$ to the next state y. At the kth transition, we incur a reward $\gamma^k g(x, u, y)$, where g is a given reward function and $0 < \gamma < 1$ is a discount factor.

A *policy* is a sequence of controls defined by a function μ mapping states into controls with $\mu(x) \in U(x)$. For infinite horizon problems, where the reward accumulates indefinitely, DP algorithms try to find the policy defined by μ that maximizes the expected discounted reward starting from an initial state x

$$J^{\mu}(x) = \lim_{N\to\infty} E\left[\sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), x_{k+1}) \mid x_0 = x\right] \quad (4)$$

The value or reward-to-go of the optimal policy starting from state x is denoted by $J^*(x)$; that is,

$$J^*(x) = \max_{\mu} J^{\mu}(x). \quad (5)$$

Markov Decision Process

When the state transition probabilities depend only on the current state and control, for a fixed policy the sequence of visited states x_k becomes a Markov chain, and the dynamic system is referred to as a Markov decision process (MDP). The existence of the Markov property enables the following recursive formulation of $J^*(x)$, known as Bellman's equation:

$$J^*(x) = \max_{u\in U(x)} \sum_{y} p_{xy}(u)\left(g(x,u,y) + \gamma J^*(y)\right), \quad \forall x. \quad (6)$$

To represent the value of a specific state-control pair (x,u), the Q-value form of Bellman's equation for the optimal policy is given as:

$$Q^*(x,u) = \sum_{y} p_{xy}(u)\left(g(x,u,y) + \gamma \max_{u'\in U(y)} Q^*(y,u')\right), \quad \forall x. \quad (7)$$

$Q^*(x,u)$ represents the value of choosing control u in state x and thereafter following the optimal policy. Once $Q^*(x,u)$ is known for all u, the optimal policy in state x is simply to select the control u that maximizes $Q^*(x,u)$.

To calculate the value function for a given policy, and so to find the optimal policy, computational-based DP methods require *a priori* knowledge of the reward function and state transition probabilities. Simulation-based DP methods do not. These methods estimate the value function for a given policy by using the policy to generate a number of simulated system trajectories and associated rewards.

Q-Learning

Watkins' Q-learning is a well-known, simulation-based DP algorithm.[9] It repeatedly updates optimal Q-value estimates, $\hat{Q}^*(x,u)$, using the following iteration:

$$\hat{Q}^*(x,u) := (1-\alpha)\hat{Q}^*(x,u) + \alpha\left(g(x,u,y) + \gamma \max_{u'\in U(y)} \hat{Q}^*(y,u')\right) \quad (8)$$

Here, the expected value in Equation 7 is replaced by a single sample, where y and $g(x,u,y)$ are generated from (x,u) by simulation. $\alpha \in (0,1]$ is a stepsize learning parameter that decreases over time. For each iteration, the control u is selected as a function of the current estimate $Q^*(x,u)$. One function often used is the ϵ-greedy algorithm. ϵ-greedy selects the control that maximizes $Q^*(x,u)$ with probability $1-\epsilon$, and a random control with probability ϵ that decreases over time. Introducing randomness into the control selection scheme ensures proper exploration of the state-control space and subsequent convergence to the optimal Q-values.

Q-learning is considered a model-free algorithm since it does not use an explicit model of the state transition probabilities and associated rewards. Model-free methods are useful to optimize control of complex systems in which the system dynamics are either difficult to model explicitly or are not sufficiently well understood, but can be generated either by simulation or by interaction with the real environment.

Neuro-Dynamic Programming

For many problems, the state-control space is too large for Q-value estimates to be stored in a lookup table representation. Consequently, neural networks have been used to approximate the value function estimates. The combination of DP and neural network (NN) function approximation methods has been referred to alternatively as connectionist reinforcement learning or neuro-dynamic programming (NDP).[8]

Generally, guarantees of convergence to the optimal Q-values that exist for the lookup table representation do not apply for an NN representation. However, experience in complex problem domains, such as elevator dispatching,[10] has shown that convergence to near-optimal performances can be achieved with careful selection of input features and other parameters.

The real power of NDP approaches lies in their generalization capabilities. By training an NDP solution on a specific problem instance, it's often possible to achieve good performance from applying the solution to an entire class of similar problems.[8] We want to develop a model-free NDP algorithm, which we can train using either offline simulation or online Web interaction, and which will subsequently gen-

eralize to yield good performance for previously unseen Web subgraphs and system parameters.

Request Controller

For the Markov property to hold, the state transition probabilities of a dynamic system must depend only on the current state and control. To model the request controller problem as a straightforward Markov decision process as above, the state must consist of the set of retrieved documents at any given time. Because the system can issue requests to check for document modifications or deletions, the control set for the state consists of both the fringe (located but not yet retrieved) and retrieved sets of documents.

A DP solution to this straightforward model of the request controller problem is computationally infeasible because the number of possible state-control pairs is exponential in the number of documents in most Web graphs of interest.

For an NDP solution, it's difficult to select features of the retrieved document set that would usefully differentiate between states for a neural network approximation of the associated Q-values. Instead, we must decompose the system's global state into "local" states, one for each document of interest. The problem's graph structure makes it natural to consider each document state as including a subset of the graph's neighboring retrieved documents.

While these local state transitions will not be strictly Markovian, we can make the transitions as Markovian as we like by extending the document's neighborhood to include an arbitrary portion of the retrieved documents. We suspect that carefully selecting state features will make it unnecessary to incorporate numerous documents in local state representations, as simulation-based DP methods have typically provided good performance even when state transitions are not strictly Markovian.[10]

For a fringe page, state features could include the average relevance score of its known parent pages, the fraction of sibling pages that have been retrieved or for which there is an outstanding request, and so on. For a retrieved page, state features could include the average relevance score for its known children pages, the fraction of children pages modified since it was last requested, and so on.

With this type of problem decomposition, how do we calculate Q-values for local state-control pairs and combine these values into a global control selection scheme? Fortunately, similar problems have been examined in the context of stochastic project scheduling problems modeled by restless bandits.

Restless Bandits

Restless bandits is a generalization of a classical project-scheduling problem known as the *multi-armed bandit* problem. In this problem, the state of project i at time k is denoted by x_k^i. In each project state, there are two controls $\mu(x_k^i) \in \{1,2\}$ representing the decisions to activate or not to activate the project, respectively. If $\mu(x_k^i)=1$, then project i receives a reward of $\gamma^k g(x_k^i, 1, x_k^{i+1})$ where the state transitions of project i form a Markov chain. In the multi-armed bandit problem, the following assumptions are made:

- States of nonactivated projects remain fixed.
- Rewards received depend only on the state of the activated project.
- Only one project can be activated at a time.

Given these assumptions, the well-known Gittins priority-index is the optimal policy.[11] This project-by-project policy calculates an index for each project separately and activates the project having the maximum index at any one time. To apply this model to the request controller problem, where each project is a document in the graph, we must eliminate the restrictions that the states of nonactivated projects are fixed, and that only one project can be active at any one time.

The restless-bandits generalization of the multi-armed bandit problem removes these restrictions and was first proposed by Whittle.[12] In this context, "restless" means that the states of nonactivated projects do not remain fixed. The restless-bandit problem is PSpace-complete.[11] PSpace-completeness most likely rules out the possibility of describing an optimal policy explicitly. Whittle, however, described a heuristic priority-index policy for a relaxed version of the restless-bandits problem, where the number of active projects was constrained to be m on average rather than fixed m. We can calculate the Whittle index on a project-by-project basis as

$$v^i\left(x_k^i\right) = Q\left(x_k^i, 1\right) - Q\left(x_k^i, 2\right), \tag{9}$$

where v is a "subsidy for passivity," and the Q-values are calculated as in Equation 7 for each project.[12] The policy at any given time is to activate the m required projects with the highest index values.

The Whittle index policy reduces to the Gittins index policy when applied to classical multi-armed bandits. While Whittle's indices for the relaxed version of the restless-bandits problem are asymptotically optimal under certain conditions,[13] these conditions do not hold, or have not been shown to hold,

for many problems. For the moment, we apply Whittle's index while noting that research into policies with stronger performance guarantees continues.[14]

Request Controller Algorithm

The request controller problem requires one addition to the restless-bandits formulation: New projects can arrive as a function of the activated projects. This addition is required because when a document is retrieved, any new links extracted are added to the request priority queue. The value of retrieving a given document is a function of that document's relevance score and also a function of the relevance scores for the subsequent retrievals that its own retrieval enables.

We define the following Q-value update rule for the discrete-time case:

$$\hat{Q}\left(x_k^i, u_k^i\right) := (1-\alpha)\hat{Q}\left(x_k^i, u_k^i\right) + \alpha\left\{\begin{array}{l} g\left(x_k^i, u_k^i, x_{k+1}^i\right) + \\ \gamma\left(\hat{Q}\left(x_{k+1}^i, u_{k+1}^i\right) + \sum_{j \in c_{k+1}(i)} \hat{Q}\left(x_{k+1}^j, u_{k+1}^j\right)\right)\end{array}\right\}, \quad (10)$$

where u_k^i is the control selected for document i and time k, and $c_{k+1}(i)$ denotes the set of i's children pages discovered as a result of requesting i at time k. This set excludes any of i's children that were previously discovered as a result of requesting another document and consists of the null set if i is not retrieved.

We define the reward function g as follows:

$$g\left(x_k^i, u_k^i, x_{k+1}^i\right) = \begin{cases} r_{k+1}(i) & \text{if } u_k^i = 1 \text{ and first time } i \text{ retrieved,} \\ \max\left[r_k(i), r_{k+1}(i)\right] & \text{if } u_k^i = 1 \text{ and } i \text{ was modified,} \\ r_k(i) & \text{if } u_k^i = 1 \text{ and } i \text{ was deleted,} \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

where $r_k(i)$ is the last relevance score calculated for document i at discrete-time k. This is not the only possible definition of the reward function; however, this definition intuitively integrates the collection's construction and maintenance.

Experimental Results

To test the request controller algorithm, we created a database of Web pages using a depth-first retrieval (to two levels) of pages pointed to by the science subdirectory of the Yahoo! directory tree. Yahoo! pages were excluded from the final database. Nekrestyanov et al. describe constructing a relevance scoring function for the subject "space" in the form of a topic filter.[6] Using this function, we calculated relevance scores for each page in the data set. We artificially constructed a start document with the 15 URLs that gave the maximum coverage of the graph, enabling approximately 140,000 pages to be reached from the start page.

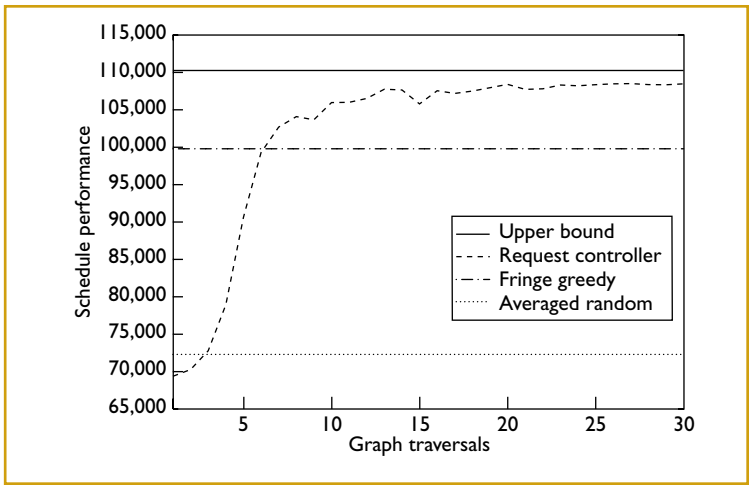

Figure 3. Performance results for different scheduling algorithms. After only 30 traversals through the test document graph, the request controller algorithm based on the restless-bandits model achieved a performance that was 96 percent of an upper bound on the highest possible performance.

Figure 3 illustrates the performance of different algorithms. The performance of the schedules generated by each algorithm was evaluated using Equation 3 where $F(C_t) = 1$ for all t. The optimal schedule for the problem without constraints—for the graph where the start node includes a link to every document in the graph—was the performance upper bound. The lower bound was the average performance of 200,000 randomly generated schedules. For comparison purposes, we also calculated the fringe greedy algorithm's performance, which simply prioritizes requests for pages by the pages' relevance score.

For the request controller algorithm, all Q-values were initially set to zero, which gave a performance worse than the average random performance for the initial traversals. Each document's local state was represented by the identity of the document responsible for its discovery. After 30 traversals through the graph, the schedule generated by the request controller algorithm achieved a performance that was 96 percent of the upper-bound performance. The algorithm required only seven traversals before it exceeded the fringe greedy algorithm's performance.

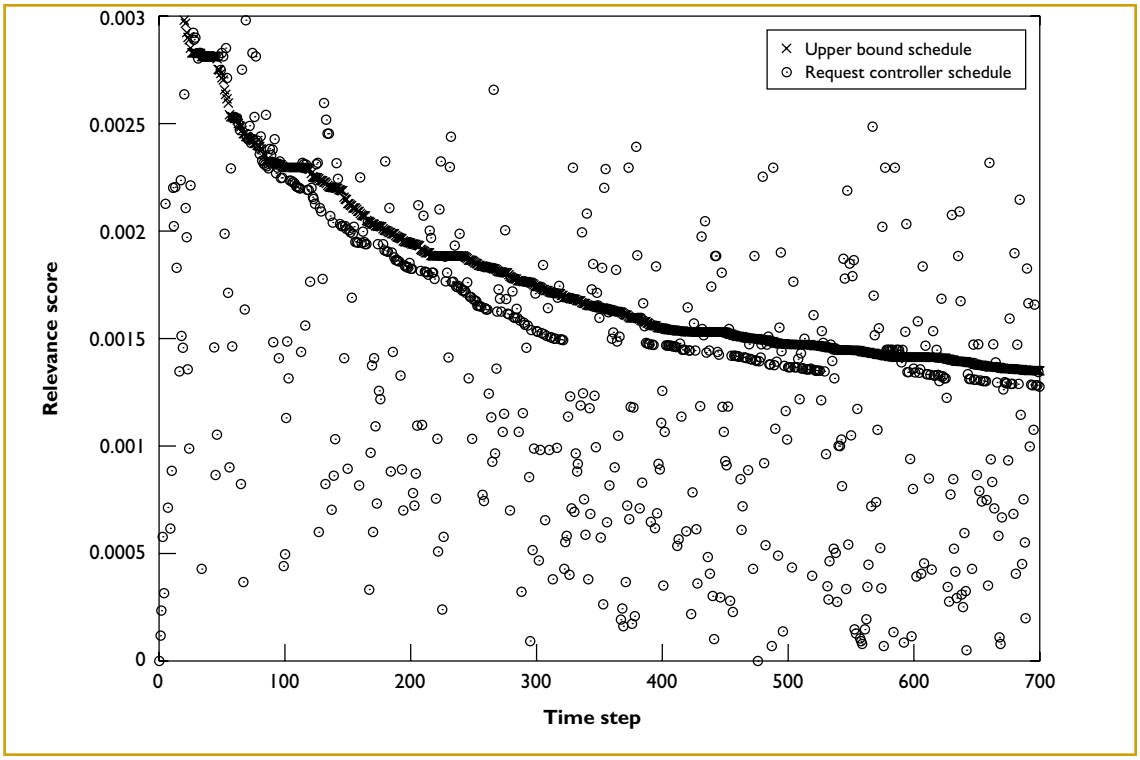

Figure 4. The relevance of the first 700 documents retrieved by upper bound and request controller schedules illustrates the request controller algorithm achieving near-upper-bound performance by not always requesting documents with the highest relevance score first.

Figures 4 and 5 provide a visual comparison between the upper bound schedule and the request controller and fringe greedy schedules for the first 700 requests. Each point in the graphs represents the relevance score of the document retrieved at a given time step.

Figures 4 and 5 show that the request controller algorithm achieves better performance than the fringe greedy algorithm by requesting some pages with low scores sooner. These pages might be what Kleinberg called "hub" pages, which have relatively little content but many links.[15] Retrieving these pages leads to the subsequent discovery and retrieval of what Kleinberg called "authoritative" or content-rich pages with higher relevance scores.

These results indicate that the request controller algorithm achieves near optimal performance using the restless bandit model.

Conclusion and Future Work

We believe this article is the first to provide a complete model of the topic-specific Web robot problem. Results of the empirical evaluation of the algorithm for a simplified instance of the model are encouraging and lead us to believe that the developed algorithm should form a good basis on which to develop NDP-based solutions for the complete model.

To evaluate the NDP algorithm for the complete model, we must first create a data testbed that contains Web documents with modifications and deletions over an extended period of time. In addition, we must develop an algorithm that can efficiently solve the fetcher controller problem. We are currently investigating and implementing solutions to these problems.

One outstanding problem of interest to researchers in other fields, particularly in project scheduling, is finding solutions to the restless-bandit problem with stronger performance guarantees than the Whittle index employed in this work. This problem has recently undergone a resurgence of interest,[14] and we look forward to applying the results of this research in the future.

References

1. S. Lawrence and C.L. Giles, "Accessibility of Information on the Web," *Nature*, vol. 400, 1999, pp. 107-109.
2. S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, 1998, pp. 107-117.
3. M. Bessonov et al., "Open Architecture for Distributed

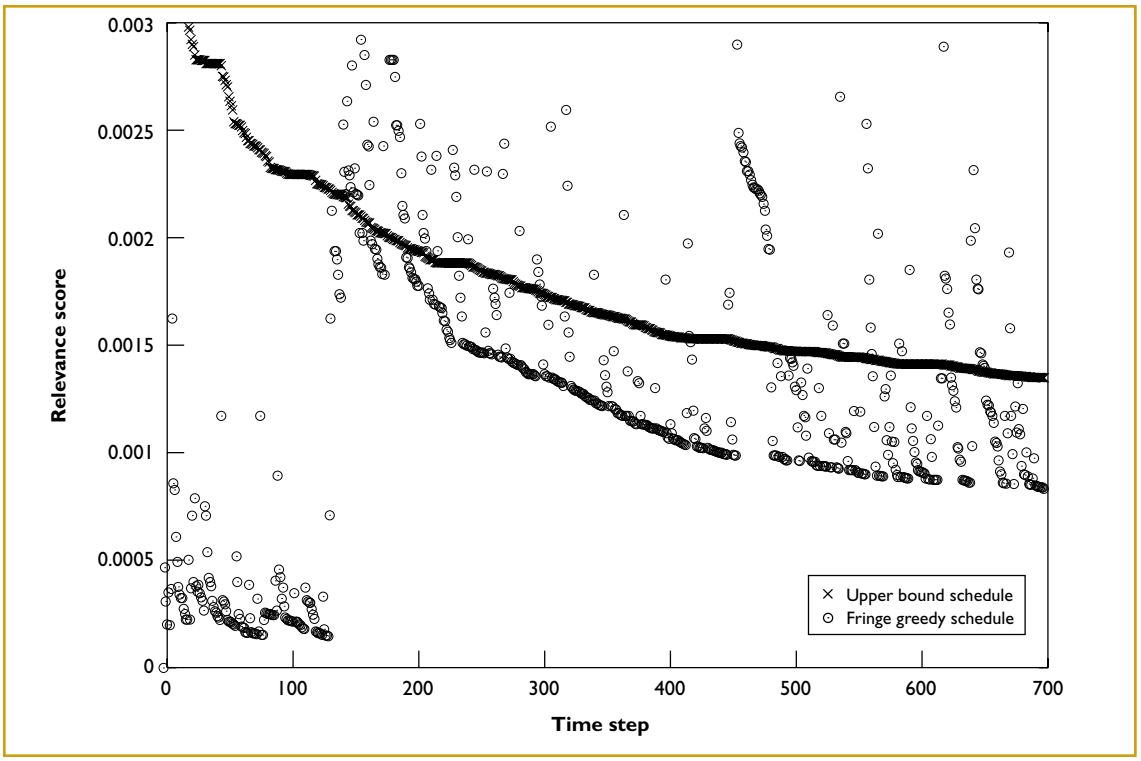

Figure 5. The relevance of the first 700 documents retrieved by upper bound and fringe greedy schedules illustrates the fringe greedy algorithm failing to achieve near upper bound performance by always requesting documents with the highest relevance score first.

Search Systems," in *Lecture Notes in Computer Science*, vol. 1597, Springer-Verlag, Berlin, 1999, pp. 55–69.

4. A. Ardö and S. Lundberg, "A Regional Distributed WWW Search and Indexing Service—the Desire Way," *Comp. Networks and ISDN Systems*, vol. 30, no. 1-7, 1998, pp. 173-183.
5. C.M. Bowman et al., "The Harvest Information Discovery and Access System," *Comp. Networks and ISDN Systems*, vol. 28, Dec. 1995, pp. 119–125.
6. I. Nekrestyanov et al., "Building Topic-Specific Collections with Intelligent Agents," in *Lecture Notes in Computer Science*, vol. 1597, Springer-Verlag, Berlin, 1999, pp. 70–82.
7. L.P. Kaelbling, M.L. Littman, and A.W. Moore, "Reinforcement Learning: A Survey," *J. Artificial Intelligence Research*, vol. 4, 1996, pp. 237–285.
8. D.P. Bertsekas and J.N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Mass., 1996.
9. C.J.C.H. Watkins, *Learning from Delayed Rewards*, PhD dissertation, Cambridge Univ., Cambridge, England, May 1989.
10. R.H. Crites and A.G. Barto, "Elevator Group Control Using Multiple Reinforcement Learning Agents," *Machine Learning*, vol. 33, Nov. 1998, pp. 235–262.
11. C.H. Papadimitriou and J.N. Tsitsiklis, "The Complexity of Optimal Queuing Network Control," *Mathematics of Operations Research*, vol. 24, May 1999, pp. 293–305.
12. P. Whittle, "Restless Bandits: Activity Allocation in a Changing World," *J. Applied Probability*, vol. 25A, 1988, pp. 287–298.
13. R. Weber and G. Weiss, "On an Index Policy for Restless Bandits," *J. Applied Probability*, vol. 27, Sept. 1990, pp. 637–648.
14. D. Bertsimas and J. Niño-Mora, "Restless Bandits, Linear Programming Relaxations and a Primal-Dual Heuristic," *Operations Research*, vol. 48, no. 1, 2000.
15. J. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM*, vol. 46, no. 5, 1999, pp. 604–632.

Ahmed Patel is a lecturer, head of the Computer Networks and Distributed Systems Research Group, and a center director of TELTEC Ireland in the Department of Computer Science, University College Dublin, Ireland. His research interests include international networking standards, network and data security, forensic computing, broadband communication systems, distributed search engines, and open distributed processing systems. He received an MS and a PhD from Trinity College, University of Dublin.

Tadhg O'Meara is a researcher and PhD candidate in the Computer Networks and Distributed Systems Research Group, University College Dublin. His research interests include distributed search engine design, neuro-dynamic programming, and Web robot design. He received a BS and an MS from the National University of Ireland, Galway.

Readers can contact the authors at {apatel,tadhg}@net-cs.ucd.ie.