A Machine Learning Architecture for Optimizing Web Search Engines

Justin Boyan, Dayne Freitag, and Thorsten Joachims
{jab,dayne,thorsten}@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
May 10, 1996*

Abstract

Indexing systems for the World Wide Web, such as Lycos and Alta Vista, play an essential role in making the Web useful and usable. These systems are based on Information Retrieval methods for indexing plain text documents, but also include heuristics for adjusting their document rankings based on the special HTML structure of Web documents. In this paper, we describe a wide range of such heuristics—including a novel one inspired by reinforcement learning techniques for propagating rewards through a graph—which can be used to affect a search engine's rankings. We then demonstrate a system which learns to combine these heuristics automatically, based on feedback collected unintrusively from users, resulting in much improved rankings.

1 Introduction

Lycos (Mauldin & Leavitt 1994), Alta Vista, and similar Web search engines have become essential as tools for locating information on the ever-growing World Wide Web. Underlying these systems are statistical methods for indexing plain text documents. However, the bulk of the Web consists of HyperText Markup Language (HTML) documents, which exhibit two kinds of structure not present in general text documents:

1. They have an *internal* structure consisting of typed text segments marked by meta-linguistic tags *(markup)*. HTML defines a set of roles to which text in a document can be assigned. Some of these roles relate to formatting, such as those defining bold and italic text. Others have richer semantic import, such as headlines and *anchors*, the text segments which serve as hyperlinks to other documents.

2. They also have an *external* structure. As a node in a hypertext, a HTML page is related to potentially huge numbers of other pages, through both the hyperlinks it contains and the hyperlinks that point to it from other pages.

Because HTML pages are more structured than general text, Web search engines enhance traditional indexing methods with heuristics that take advantage of this structure. It is by no means clear how to integrate such heuristics most effectively, however.

Paper Overview

In the following section we describe our prototype Web-indexing system, called **LASER**, and outline its heuristics for exploiting the internal and external structure of the Web. In the section entitled **Automatic Optimization**, we describe how the parameters for combining these heuristics are automatically tuned based on system usage. Finally, we present and discuss our first empirical results with the system.

2 LASER

LASER, a Learning Architecture for Search Engine Retrieval, is a system designed to investigate the applicability of Machine Learning methods to the indexing of Web pages. From a user's perspective, much of LASER's functionality is identical to that of other popular Web search engines (see Figure 1). The user enters unstructured keyword queries, which LASER matches against its index of pages, returning abstracts of and links to the 60 pages matching the query most closely. From this page of search results, the user can proceed to any of the abstracted pages or enter a new query.

LASER's retrieval function is based on the TFIDF vector space retrieval model (Salton 1991). In this model documents and queries are represented as vectors of real numbers, one for each word; documents and queries with similar contents are transformed into similar vectors. LASER uses an inner product similarity

*To appear in: AAAI Workshop on Internet-Based Information Systems, Portland, Oregon, 1996.

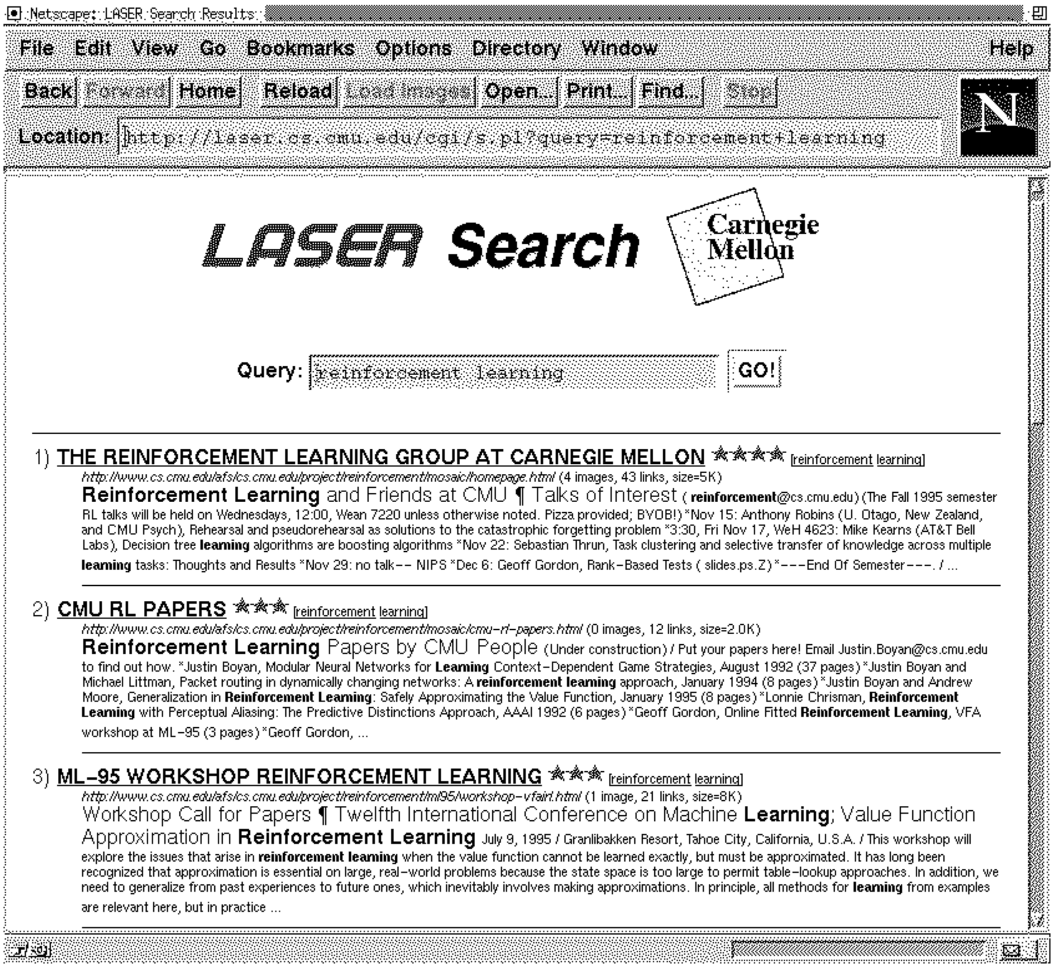

Figure 1: The LASER Interface. Our prototype system indexes approximately 30,000 hypertext documents available from the CMU Computer Science Department Web server.

metric to compare documents with a query. Typically, the value of a word depends both on its frequency in the document under consideration and its frequency in the entire collection of documents. If a word occurs more frequently in a particular document than in the collection as a whole, then it is considered salient for that document and is given a high score. In its simplest form, TFIDF assigns to each word a score proportional to its frequency in the document (*term frequency* or *TF*) and a decreasing function of the number of documents it occurs in overall (*inverse document frequency* or *IDF*).

LASER's retrieval function, based on this model, offers a number of parameters which influence the rankings it produces. The parameters affect how the retrieval function responds to words in certain HTML fields (like headlines), how hyperlinks are incorporated, how to adjust for partial-word matches or query-term adjacency, and more: altogether, there are 18 real-valued parameters.[1] Using a particular parameter setting makes it possible to pick a certain retrieval function from the family of functions LASER offers. In this way, the retrieval function can be adjusted to the different characteristics of various document collections and user groups.

2.1 Using HTML Formatting in Retrieval

Most Web pages are written in HTML. HTML is a markup language which allows the designer of a page to assign certain semantics to parts of a document and to control the layout. The designer can specify, for example, the title of a document, hierarchies of headlines and hyperlinks, and character formats such as boldfac-

[1] A listing of the parameters LASER uses, in the form of a function for calculating document scores, can be found in the Appendix.

ing.

LASER makes use of the structure HTML imposes on documents. For example, one parameter governs to what extent words in the title of a document should receive stronger indexing weight than words near the end of a document. LASER has parameters for weighting words in the following HTML fields:

- TITLE
- H1, H2, H3 (headlines)
- B (bold), I (italics), BLINK
- A (underlined anchor text)

The parameters for these HTML tags are simply multiplicative factors for the "term frequency" of words within their scope.

2.2 Incorporating Hypertext Links

Unlike most other document collections, Web pages are part of a hypertext graph. For retrieval it might be useful not only to look at a document in isolation, but also to take its neighboring documents into account.

The approach we took is motivated by an analogy to reinforcement learning as studied in artificial intelligence (Barto, Bradtke, & Singh 1995). Imagine that an agent searching for information on the Web can move from page to page only by following hyperlinks. Whenever the agent finds information relevant to its search goal, it gets a certain amount of reward. Reinforcement learning could be used to have the agent learn how to maximize the reward it receives, i.e. learn how to navigate to relevant information.

The idea, then, is to have LASER rank highly pages that would serve as good starting points for a search by such an agent. Good starting points are pages from which it is easy to reach other pages with relevant information. We conjecture that these pages are relevant to a query even if they do not contain much relevant information themselves, but just link to a number of relevant documents.

Hyperlinks are incorporated as follows. First, given a query q the retrieval status values $\mathrm{rsv}_0(\mathrm{q},\mathrm{d})$ are calculated for each page d in the collection independently, based on the HTML-specific TFIDF parameters described above. In reinforcement-learning terms, these values represent the "immediate reward" associated with each page. Then, LASER propagates the rewards back through the hypertext graph, discounting them at each step, by *value iteration* (Bellman 1957):

$$\mathrm{rsv}_{\mathrm{t+1}}(\mathrm{q},\mathrm{d}) = \mathrm{rsv}_0(\mathrm{q},\mathrm{d}) + \gamma \sum_{\mathrm{d}' \in \mathrm{links}(\mathrm{d})} \frac{\mathrm{rsv}_{\mathrm{t}}(\mathrm{q},\mathrm{d}')}{|\mathrm{links}(\mathrm{d})|^{\nu}} \quad (1)$$

γ is a discount factor that controls the influence of neighboring pages, and links(d) is the set of pages referenced by hyperlinks in page d. This dynamic-programming update formula is applied repeatedly for each document in a subset of the collection. This subset consists of the documents with a sigificant rsv_0, and it also includes the documents that link to at least one of those. After convergence (in practice, 5 iterations), pages which are n hyperlinks away from document d make a contribution proportional to γ^n times their retrieval status value to the retrieval status value of d.

Two parameters to LASER influence the behavior of this mechanism: one is γ, and the other, $\nu \in [0,1]$, controls the normalization of the denominator in Formula 1 in a range from |links(d)| down to 1. Altogether, our retrieval status function has 18 parameters; the score assigned to document d in the context of query q is computed by $\mathrm{rsv}_5(\mathrm{q},\mathrm{d})$ as detailed in the Appendix.

3 Automatic Optimization

The 18 numerical parameters of LASER's retrieval function allow for a wide variety of search engine behavior, from plain TFIDF to very complex ranking schemes. Qualitatively, different retrieval functions produce markedly different rankings (see Table 1). Our goal is to analyze system usage patterns to (1) quantify these differences, and (2) automatically optimize the parameter settings.

3.1 Measuring Search Engine Performance

In order to keep the system interface easy to use, we made a design decision *not* to require users to give explicit feedback on which search hits were good and which were bad. Instead, we simply record which hits people follow, e.g. "User searched for 'vegetarian restaurant' and clicked on *Restaurant Reviews* and *Eating 'Indian' in Pittsburgh*." Because the user gets to see a detailed abstract of each hit (see Figure 1), we believe that the hits actually clicked by the user are highly likely to be relevant.

A good retrieval function will obey the probability ranking principle (van Rijsbergen 1979). This means it places documents which are most likely to be relevant to the user's query near the top of the hit list. To evaluate a retrieval function f's performance on a single query q, we simply take the mean ranking according to f of all documents the user followed. (Example scores are shown in Table 1.) We then define the overall performance of retrieval function f to be the average of its performance over all queries in the database. In

standard TFIDF	using HTML structure; hand-tuned parameters
1. Vegetarian Chili Recipes	**1. Restaurant Reviews**
2. Vegetarian Recipes	**2. Eating "Indian" in Pittsburgh**
3. Eating "Indian" in Pittsburgh	3. A List of Food and Cooking Sites
4. Restaurant Reviews	4. Duane's Home Page & Gay Lists
5. Greek Dishes	5. Eating & Shopping Green in Pittsburgh
6. Focus on Vegetarian	6. Living Indian in Pittsburgh
7. For the Professional Cook	7. For the Professional Cook
SCORE: 3.5	SCORE: 1.5
simple count of query terms	**using HTML structure; automatically-learned params**
1. Collection: Thai Recipes	**1. Eating "Indian" in Pittsburgh**
2. Food Stores Online	**2. Restaurant Reviews**
3. A List of Food and Cooking Sites	3. A List of Food and Cooking Sites
4. Cookbook of the Year	4. Vegetarian Chili Recipes
5. Collection: Tofu	5. For the Professional Cook
6. Eating "Indian" in Pittsburgh	6. Eating & Shopping Green in Pittsburgh
... **16. Restaurant Reviews**	7. Vegetarian Recipes
SCORE: 11	SCORE: 1.5

Table 1: Rankings produced by four different retrieval functions in response to the query "vegetarian restaurant." Supposing that the user had clicked on the *Eating "Indian" in Pittsburgh* and *Restaurant Reviews* pages, these retrieval functions would be scored as shown.

symbols:

$$Perf(f) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{|D_i|}\sum_{j=1}^{|D_i|}\mathrm{rank(f, Q_i, D_{ij})} \qquad (2)$$

where $Q_1 \ldots Q_{|Q|}$ are the queries in our database and D_i is the set of documents the user followed after posing query Q_i. The input used by this performance method is clearly noisier and more biased than that used in methods based on precision-recall (van Rijsbergen 1979), which employ exhaustive relevance feedback information assigned manually by experts.

In practice, the user's choice of hits to follow is strongly biased toward documents appearing early in the hit list—regardless of the quality of retrieval function used. Users rarely have the patience to scroll through pages and pages of hits. Thus, when evaluating performances of new retrieval functions using our collected database, we attempt to equalize these "presentation biases." We do this by evaluating *Perf* on a subsample Q' of our query database, where Q' is constructed to contain an equal number of queries from each different presentation bias; or alternatively, we weight each query Q_i so as to give equal total weight to each presentation bias.

3.2 Optimization Method

Given our parametrization of the space of retrieval functions and our metric for evaluating a retrieval function's performance, we can now pose the problem of finding the best retrieval function as a problem of function optimization: find the parameter vector $\vec{p}$ minimizing $Perf(f_{\vec{p}})$.

The calculation of *Perf* is based on averages of discrete rankings, so we expect it to be quite discontinuous and probably riddled with local minima. Thus, we chose to apply a global optimization algorithm, simulated annealing. In particular, we applied the "modified downhill simplex" variant of simulated annealing, as described in (Press *et al.* 1992).

Because we calculate *Perf* from only a fixed subsample of queries, aggressive minimization introduces the danger of overfitting; that is, our converged parameter vector $\vec{p}$ may exploit particular idiosyncracies of the subsample at the expense of generalization over the whole space. To guard against overfitting, we use early stopping with a holdout set, as is frequently done in neural network optimization (Morgan & Bourlard 1990), as follows:

1. We consider the sequence of parameter vectors which are the "best so far" during the simulated annealing run. These produce a monotonically decreasing learning curve (see, for example, Figure 2).

2. We then evaluate the performance of each of these vectors on a separate holdout set of queries. We smooth the holdout-set performance curve and pick

↓ f used for presentation	count	TFIDF	hand-tuned
count	**6.26**± 1.14	46.94±9.80	28.87±7.39
TFIDF	54.02±10.63	**6.18**±1.33	13.76±3.33
hand-tuned	48.52± 6.32	24.61±4.65	**6.04**±0.92
Overall Performance	**36.27**± 4.14	**25.91**±3.64	**16.22**±2.72

Table 2: Performance comparison for three retrieval functions as of March 12, 1996. Lower numbers indicate better performance. Rows correspond to the indexing method used by LASER at query time; columns hold values from subsequent evaluation with other methods. Figures reported are means ± two standard errors (95% confidence intervals).

its minimum; the parameter setting thus chosen is the final answer from our optimization run.

Each evaluation of $Perf(f_{\vec{p}})$ on a new set of parameters is quite expensive, since it involves one call to the search engine for each query in Q'. These evaluations could be sped up if Q' were subsampled randomly on each call to $Perf$; however, this adds noise to the evaluation. We are investigating the use of stochastic optimization techniques, which are designed for optimization of just this type of noisy and expensive objective function (Moore & Schneider 1996).

4 Empirical Results

LASER has been in operation since February 14, 1996. The system currently indexes a document database consisting of about 30,000 pages served by the CMU Computer Science Department web server, `www.cs.cmu.edu`. The system is available for use by the CMU community from the department's local home page, `http://www.cs.cmu.edu/Web/SCS-HOME.html`. (We are considering plans for larger indexes and wider release.)

4.1 Validity of Performance Measure

We first ran an experiment to determine whether our performance function could really measure significant differences between search engines, based only on unintrusive user feedback. We manually constructed three retrieval functions:

simple-count scores a document by counting the number of query terms which appear in it;

standard-TFIDF captures word relevance much better but does not take advantage of HTML structure; and

hand-tuned includes manually-chosen values for all 18 parameters of our HTML-specific retrieval function.

From February 14 through March 12, we operated LASER in a mode where it would randomly select one of these three retrieval functions to use for each query. During this time LASER answered a total of 1400 user queries (not including queries made by its designers). For about half these queries, the user followed one or more of the suggested documents.

We evaluated $Perf(f)$ for each engine according to Equation 2. The results, shown in the bottom row of Table 2, indicate that our performance metric does indeed distinguish the three ranking functions: hand-tuned is significantly better than TFIDF, which in turn is significantly better than simple-count.

The first three rows of Table 2 break down the performance measurement according to which retrieval function generated the original ranking seen by the user. The presentation bias is clear: the diagonal entries are by far the smallest. Note that the diagonal entries are not significantly different from one another with the quantity of data we had collected at this point.

However, we do see significant differences when we average down the columns to produce our full performance measurements in row four. Moreover, ranking the three methods according to these scores produces the expected order. We take this as evidence that our performance measure captures to some extent the "goodness" of a retrieval function and can serve as a reasonable objective function for optimization.

4.2 Optimization Results

To date, we have had time to run only one optimization experiment, so the results of this section should be considered preliminary. Our goal was to minimize $Perf(f_{\vec{p}})$, thereby producing a new and better ranking function.

For efficiency in evaluating $Perf$, we let Q be a fixed subsample of 150 queries from our full query database, 50 from each presentation bias. To make the search space more tractable, we optimized over only a 10-dimensional projection of the full 18-dimensional parameter space. These 10 parameters still allowed for

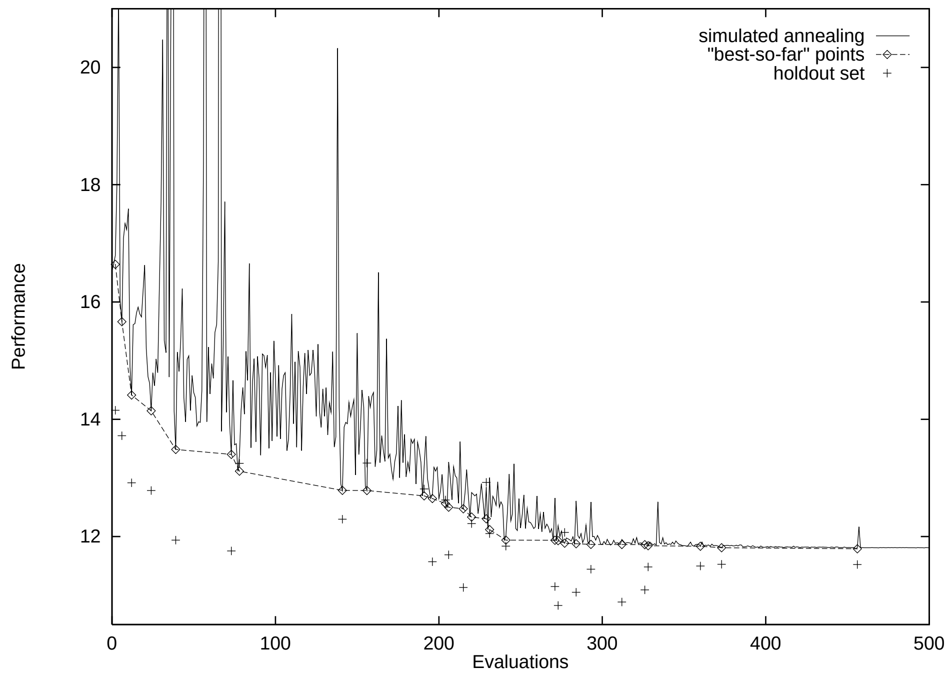

Figure 2: Optimization of search engine performance by simulated annealing. $Perf(f)$ is plotted for each of the 500 different parameter settings explored. Evaluations on a separate holdout set are used to prevent overfitting.

tuning of such heuristics as title and heading bonuses, query-word adjacency bonus, partial-match penalty, document length penalty, near-top-of-page bonus, and γ, our hypertext discount factor.

As described above, we ran simulated annealing to find a new, optimal set of retrieval function parameters. Simulated annealing's own parameters were set as follows: temperature at evaluation #i $= 10.0 \cdot 0.95^{i/2}$, and initial stepsize $=$ 10% of the legal range for each dimension. This run converged after about 500 evaluations of $Perf(f_{\vec{p}})$ (see Figure 2). Using the early-stopping technique, we chose the parameter setting at evaluation #312 as our final answer.

Compared to our hand-tuned parameter setting, the learned parameter setting gave more weight to title words, underlined anchor text, and words near the beginning of a document. Surprisingly, it set γ (our graph-propagation discount factor) almost to 0. Installing the new retrieval function into our search engine interface, we found it produced qualitatively good rankings (e.g., refer back to Table 1).

From March 26 through May 6, LASER generated its rankings with the new retrieval function half the time and with our hand-tuned retrieval function half the time. The cumulative results are shown in Table 3. According to our overall performance metric, the hand-tuned and learned retrieval functions both significantly outperform count and TFIDF, but do not differ significantly from one another.

However, the diagonal entries, which reflect actual use of the system, provide some indication that the learned function is an improvement: with 88% confidence, the learned retrieval function's value of 4.87 ± 0.56 is better than our hand-tuned function's value of 5.33 ± 0.57. If this trend continues, we will be satisfied that we have successfully learned a new and better ranking scheme.

5 Related Work

Many retrieval engines have been developed to index World Wide Web pages. Descriptions of some can be found in (Mauldin & Leavitt 1994) and (Pinkerton 1994). These retrieval engines make use of the internal structure of documents, but they do not incorporate hyperlinks. Other researchers have focused on retrieval using hypertext structure without making use of the internal structure of documents (Savoy 1991; Croft & Turtle 1993).

Automatic parameter optimization was previously proposed by (Fuhr *et al.* 1994) as well as (Bartell, Cottrell, & Belew 1994). Both approaches differ from LASER's in that they use real relevance feedback data. LASER does not require relevance feedback assignment by the user; it uses more noisy data which can

↓ f used for presentation	count	TFIDF	hand-tuned	learned
count	**6.33**± 1.13	48.19±9.90	30.62±7.75	32.60±7.97
TFIDF	55.43±10.32	**6.05**±1.25	13.31±3.14	8.22±2.12
hand-tuned	50.55± 4.68	21.34±2.98	**5.33**±0.57	8.95±1.58
learned	47.36± 4.99	13.14±2.21	7.22±0.95	**4.87**±0.56
Overall Performance	39.92± 3.11	22.18±2.66	14.12±2.11	13.66±2.10

Table 3: Cumulative performance comparison for four retrieval functions as of May 6, 1996. The data is reported in the same format as in Table 2.

be collected unintrusively by observing users' actions.

6 Conclusions and Future Work

Initial results from LASER are promising. We have shown that unintrusive feedback can provide sufficient information to evaluate and optimize the performance of a retrieval function. According to our performance metric, an index which takes advantage of HTML structure outperforms a more traditional "flat" index. Furthermore, we have begun to collect results demonstrating that it is possible to automatically improve a retrieval function by learning from user actions, without recourse to the intrusive methods of relevance feedback.

There are many directions for further research, which we see as falling into three general areas:

retrieval function parametrization LASER currently offers 18 tunable parameters for combining heuristics into a retrieval function, but certainly many other heuristics are possible. For example, we would like to further refine our method for incorporating hyperlinks. We are also planning to include per-document popularity statistics, gathered from regular LASER usage, into the relevance function. If a document is always skipped by LASER users, the system should learn to punish that document in the rankings.

evaluation metrics While our performance function has an appealing simplicity, and agrees with our qualitative judgments on the three search engines of Table 2, we cannot defend it on theoretical grounds. A metric derived directly from the probabilistic ranking principle (van Rijsbergen 1979), for example, would allow us to make stronger claims about our optimization procedure. Another alternative is to implement a cost function over rankings, where the cost increases with the number of irrelevant links (i.e., those which the user explicitly skipped over) high in the ranking. It is not clear whether this is a useful metric, or even how to decide among these alternatives.

On a related issue, we have documented a pronounced tendency for users to select links that are high in the rankings, no matter how poor the index, resulting in "presentation bias." This complicates the problem of evaluating new retrieval functions offline during optimization, since our query database will strongly bias the retrieval parameters toward those used for the original presentation. We have an *ad hoc* method for compensating for this effect, but would be interested in more principled approaches.

optimization As mentioned in Section 3.2, we plan to investigate the use of stochastic optimization techniques, in place of simulated annealing, for optimizing the parameter settings. There is also an interesting possibility for "lifetime learning." We would like to see how the system improves over time, iteratively replacing its index with a new and improved one learned from user data. We can only speculate about the trajectory the system might take. There is the possibility of an interesting kind of feedback between the system and its users; as the system changes its indexing behavior, perhaps the users of the system will change their model of it and use it somehow differently from at first. Is there a globally optimal parameter setting for LASER? It may be that, given the presentation bias and the possibility of drifting patterns of use, its parameters would never settle into a stable state.

Acknowledgments

We would like to thank Tom Mitchell and Andrew Moore for the computational and cognitive resources they shared with us for these experiments. Thanks, too, to Darrell Kindred for counseling us on indexing the local Web. Finally, thanks to Michael Mauldin, author of the Scout retrieval engine which we used as a basis for our own.

Appendix A Parametric Form of Retrieval Function

$$
\begin{aligned}
\mathrm{rsv}_{t+1}(\mathrm{q},\mathrm{d}) &= \mathrm{rsv}_0(\mathrm{q},\mathrm{d}) + \$\mathrm{gamma} \sum_{\mathrm{d}'\in \mathrm{links}(\mathrm{d})} \frac{\mathrm{rsv}_t(\mathrm{q},\mathrm{d}')}{|\mathrm{links}(\mathrm{d})|^{\$\mathrm{nu}}} \\
\mathrm{rsv}_0(\mathrm{d},\mathrm{q}) &= multihit(q,d) \cdot \sum_{i=1}^{|q|} \sum_{j=1}^{|d|} [q_i = d_j] \cdot \frac{qweight(i,q_i,d_j)}{|q|} \cdot \frac{dweight(j,q_i,d_j)}{|d|^{\$doclen\_exp}} \cdot adjacency(q_{i-1},d_{j-1}) \\
qweight(i,q_i,d_j) &= \frac{1}{i}^{\$query\_pos\_exp} \cdot idf(q_i) \cdot (1 + isfullmatch(q_i,d_j) \cdot \$fullmatch\_factor \\
&\qquad + ispartmatch(q_i,d_j) \cdot \$partmatch\_factor) \\
dweight(j,d_j) &= idf(d_j) \cdot (1 + in\_h1\_headline(d_j) \cdot \$h1\_factor \\
&\qquad + in\_h2\_headline(d_j) \cdot \$h2\_factor \\
&\qquad + in\_h3\_headline(d_j) \cdot \$h3\_factor \\
&\qquad + in\_title(d_j) \cdot \$title\_factor \\
&\qquad + in\_bold(d_j) \cdot \$bold\_factor \\
&\qquad + in\_italics(d_j) \cdot \$italics\_factor \\
&\qquad + in\_blink(d_j) \cdot \$blink\_factor \\
&\qquad + in\_anchor(d_j) \cdot \$anchor\_factor \\
&\qquad + \frac{\$toppage\_factor}{\log(j + \$toppage\_add)}) \\
adjacency(q_{i-1},d_{j-1}) &= [q_{i-1} \neq d_{j-1}] + [q_{i-1} = d_{j-1}] \cdot \$adjacency\_factor \\
multihit(q,d) &= (number\_of\_words\_in\_q\_that\_occur\_in\_d)^{\$multihit\_exp}
\end{aligned}
$$

References

Bartell, B.; Cottrell, G.; and Belew, R. 1994. Optimizing parameters in a ranked retrieval system using multi-query relevance feedback. In *Proceedings of Symposium on Document Analysis and Information Retrieval (SDAIR)*.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

Bellman, R. 1957. *Dynamic Programming.* Princeton University Press.

Croft, B., and Turtle, H. 1993. Retrieval strategies for hypertext. *Information Processing and Management* 29(3):313–324.

Fuhr, N.; Pfeifer, U.; Bremkamp, C.; Pollmann, M.; and Buckley, C. 1994. Probabilistic learning approaches for indexing and retrieval with the TREC-2 collection. In *The Second Text Retrieval Conference (TREC-2).* National Institute of Standards and Technology.

Mauldin, M., and Leavitt, J. 1994. Web agent related research at the Center for Machine Translation. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval (SIGNIDR-94).*

Moore, A., and Schneider, J. 1996. Memory-based stochastic optimization. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Neural Information Processing Systems 8.* MIT Press.

Morgan, N., and Bourlard, H. 1990. Generalization and parameter estimation in feedforward nets: Some experiments. In Touretsky, D. S., ed., *Neural Information Processing Systems 2*, 630–637. Morgan Kaufmann.

Pinkerton, B. 1994. Finding what people want: Experiences with the WebCrawler. In *Second International WWW Conference.*

Press, W.; Teukolsky, S.; Vetterling, W.; and Flannery, B. 1992. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second edition.

Salton, G. 1991. Developments in automatic text retrieval. *Science* 253:974–979.

Savoy, J. 1991. Spreading activation in hypertext systems. Technical report, Université de Montréal.

van Rijsbergen, C. 1979. *Information Retrieval.* London: Butterworths, second edition.