Exploring Browser Design Trade-offs Using a Dynamical Model of Optimal Information Foraging

Peter Pirolli Xerox PARC 3333 Coyote Hill Road Palo Alto, CA 94304 pirolli@parc.xerox.com

ABSTRACT

Designers and researchers of human-computer interaction need tools that permit the rapid exploration and management of hypotheses about complex interactions of designs, task conditions, and user strategies. Dynamic programming is introduced as a such a tool for the analysis of information foraging technologies. The technique is illustrated in the context of the Scatter/Gather text clustering browser. Hypothetical improvements in browser speed and text clustering are examined in the context of variations in task deadlines and the quality of the document repository. A complex and non-intuitive set of tradeoffs emerge from even this simple space of factors, illustrating the general utility of the approach.

Keywords

Dynamic programming, information foraging, Scatter/Gather, user models.

INTRODUCTION

Surveys of users of the World Wide Web (WWW) find that the two most frequently reported problems are poor speed of access and failure to find information that is known to be available (e.g., Pitkow & Kehoe, 1996). Designers of browsers for such large and rapidly-growing hypermedia repositories will naturally be concerned with alleviating those problems. Like all complex design problems, however, there will be many interacting constraints and trade-offs in the space of potential designs. These design trade-offs may also vary according to the space of conditions that will be faced by potential users. Furthermore, one may want to predict some of the effects of these designs on user strategies.

For designers of user interfaces, such as browsers, it might be helpful to have techniques that allow one to explore various complex "what-if" design scenarios. For instance, what if system algorithms are made faster as opposed to more accurate? What if presentations are made more informative but slower to read? What if the user has unlimited time as opposed to a hard deadline? What if the user is faced with a repository rich with relevant information vs a poor one? This paper presents a modeling technique for exploring a space of human-computer interaction designs. It is a dynamical modeling technique that was initially suggested in the context of Information Foraging Theory (Pirolli & Card, 1995). Here I present a more elaborate description of the technique and its application to exploring the design space for a browser for very large text repositories. It seems likely that these techniques could be extended to other browsers, such as those for the World Wide Web.

Basically, the approach requires that the analyst find an abstract representation of the different states of interaction, such as the state of a browser display, and the different changes that can be made from state to state, such as the changes that result from user actions. This defines an abstract state space representing the possible paths that human-computer interaction may take. One also must have some method for assigning costs and values to different states and moves. In the example I describe below, the values are the expected number of relevant documents that will be encountered while browsing, and the costs are just the amounts of user time involved. Different state spaces, with different costs and values, are used to represent alternative interfaces. I then use a technique called dynamic programming (Bertsekas, 1995) to perform an evaluation of the different interfaces. Conceptually, it does this by searching through the different possible paths of human-computer interaction, evaluating the costs and values of different paths, and finding the best paths. In this manner, one can find the best-case performance of a user interface. This is the fitness of the interface.

One goal of this paper is to provide a more thorough introduction to the dynamic programming analysis of information foraging by application to a concrete example. For the engineer/designer this technique is proposed as a sorely needed tool for the rapid exploration of "what-if" variations in complex designs. For the researcher, it is a technique for exploring and generating hypothesis about the interaction of design trade-offs, task conditions, and user strategies. It should be viewed as a technique for



Figure 1. The Scatter/Gather cluster display window.

making well-informed hypothesis about complex design trade-offs. The validity of the technique will depend on many factors that the designer may wish to check empirically.

A dynamic programming analysis of information foraging is presented for a browser that clusters large-scale text collections, called Scatter/Gather (Cutting, Karger, & Pedersen, 1993; Cutting, Karger, Pedersen, & Tukey, 1992; Pirolli, Schank, Hearst, & Diehl, 1996) The interaction space of the baseline model is based on empirical data (Pirolli & Card, 1995; Pirolli et al., 1996). Variations on this model consider the simulated effects of, and interactions among, (a) different deadlines, (b) different amounts of available relevant information, (c) possible improvements in interaction time costs, and (d) possible improvements clustering of relevant information.

THE SCATTER/GATHER BROWSER

The Scatter/Gather browser (Cutting et al., 1993; Cutting et al., 1992; Pirolli et al., 1996) uses the clustering of documents as the basis of a browser suitable for large numbers of documents (Figure 1). Each of the ten subwindows in Figure 1 represents a cluster of documents. Each subwindow presents a cluster digest, which contains topical words and the titles of the most typical documents in that cluster. The clustering and cluster digests are computed by automatic means based on the texts themselves. The user may *gather* clusters of interest by pointing and selecting buttons above each cluster. On command, the system will pool together the documents in those clusters, then automatically *scatter* that subcollection into another set of clusters. The user may repeatedly scatter then gather clusters, moving from very large cluster collections to very small cluster collections. Eventually the user may display all the titles of documents in one or more clusters, then select individual documents to read.

In studies (Pirolli & Card, 1995; Pirolli et al., 1996) Scatter/Gather was applied to the Tipster collection of about 750,000 documents created for the TREC text retrieval conference (Harman, 1993). Standard information retrieval tasks (queries) have been defined on it together with lists of known relevant and non-relevant Tipster documents, as judged by experts. We studied this version of Scatter/Gather under experimental conditions (Pirolli & Card, 1995; Pirolli et al., 1996) in which the general objective for users was to find as many bibliographic references as possible relevant to a set of the TREC queries. This forms the basic starting point for our dynamic programming model.



Figure 3. A portion of the Scatter/Gather state space.



Figure 2. A schematic of a simple path of Scatter/Gather interaction.

DYNAMIC INFORMATION FORAGING MODELS

The dynamic modeling approach to information foraging taken here was inspired by similar approaches in the study of the ecology of animal behavior (Mangel & Clark, 1988). A more technical and mathematical treatment of the current model is provided in Pirolli and Card (1997). The dynamic optimization techniques used here are just a small example of an extensive set of such models (Bertsekas, 1995).

State Space of Interaction

Figure 2 gives a schematic overview of a path of humancomputer interaction using the Scatter/Gather browser. Each icon in Figure 2 represents an interaction state involving one of the two main kinds of display windows. The Scatter/Gather display window presents clusters that the user may gather. Eventually, a Display Titles window is used to display the titles of documents in clusters chosen by the user, and the user scans these seeking relevant ones. The boxes beneath the icons schematically represent information about the interaction states. The sharpcornered boxes represent Scatter/Gather states and the round-edged boxes represent Display Titles states. The boxes in Figure 2 contain a the following subset of state information used in the models:

- Time (*T*) in seconds from the start of the simulated foraging task. For the Scatter/Gather states, these times record the point at which the window is displayed and the next user action begins. For the Display Titles states these times record the point at which the window has been displayed and the user has completed scanning and selecting titles. If this display scanning would go beyond the task deadline (720 seconds in the Figure 2 example), then the state's time is set to the deadline.
- Total number of documents (*N*) in the part of the collection displayed in the current window.
- The estimated number of relevant documents (*R*) in the collection displayed in the current window
- The best-case (optimal) number of relevant documents (*G*) that can be gained from the current state.

The number label on the arrow between the icons in Figure 2 indicates the number of clusters gathered at a state along the path (the model assumes that the best clusters are chosen), or a "D" label indicates that the chosen clusters were displayed in a Display Titles window.

The path in Figure 1 models a user who starts out by gathering two clusters at time T = 0 seconds, and scattering these into a new Scatter/Gather state that appears at time T = 52 seconds. One cluster from this state is gathered and displayed in a Display Title window, and scanned and selected until the deadline time T = 720 seconds. One can note in Figure 2 that the total number of documents as well as the number of relevant documents are reduced as one moves from state to state. However, one can also see that the proportion of relevant documents R/N is increasing. The manner in which the expected G is computed is at the

core of the dynamic programming algorithm and is discussed below.

The collection of all the states achievable along all the paths of interaction from some given start state define a state space. Rather than the single path shown in Figure 2, a subset of the paths are shown in Figure 3 (the icons are now omitted). Only a small portion of the full state space is displayed in Figure 3 for the purposes of illustration. Only the first few steps in the interaction space are shown, only alternative moves along the best paths are displayed, and only three of the alternatives are shown. The full space gets explosively large as more alternatives are added,¹ and this is often the main computational drawback to using dynamic state-space models.

Dynamic Programming Approach

Imagine that one could, however, generate all possible paths and isolate all the possible interaction states that a user could get to by the task deadline (this may be an infinite set of end states). One could evaluate these states This value might be any to determine their value. resource, but in our example it is the number of relevant articles that can be collected by the task deadline. Now imagine that one could take one step backward from the end states. From these penultimate states, the optimum step would be the one that goes to the highest valued end state, which is known from the evaluation of end states. So, the optimum value of the penultimate states can be calculated by tracing backward from the end states. Generalizing this process, one may iterate the process backwards from states to prior states until one gets back to the starting state.

To summarize, the dynamic programming approach involves defining a state space and an optimization criterion. In the current example, the state space is defined by the representation of states, a particular starting state, a set of feasible strategies and actions, and the state dynamics produced by those state-changing actions. If the optimization problem is formulated in an appropriate and way (Bertsekas, 1995), tractable then dynamic programming finds the sequence of states and action optimizes choices that the specified criterion. Conceptually, the dynamic programming optimization technique finds the value of end states (at a task deadline)

and works backward along the interaction paths to label states with their optimal gains. In practice, there are many ways to implement dynamic programming (Bertsekas, 1995).

Technical Summary of Dynamic Programming²

Let X(t) = x be a state variable representing the state of interaction at time *t*. As described in Pirolli and Card (1997) we use a multi-dimensional vector to represent the Scatter/Gather states. The components of the X(t) vector in this example would include *N*, *R*, *T*, *G*, as well as other state attributes.

There will also be a set of state-change operators $\delta_i(X(t))$, that produce some new state, $X(t + C) = \delta_i(X(t))$. For instance, the user actions of gathering and scattering clusters in Scatter/Gather are examples of such state-change operators. The time cost of the operator will be *C*, and its value may be state-dependent. For instance, the time cost of displaying titles in Scatter/Gather depends on the total documents, *N*, and relevant documents, *R*.

For our current definition, let time *t* be indexed in seconds remaining to deadline. For a task with a deadline of 720 seconds: at the beginning of the task, t = 720 and at deadline t = 0 seconds. Using the foraging terminology of Mangel and Clark (1988), we construct a *fitness* evaluation function $\phi(X(t))$ for the final end states. For the current Scatter/Gather example, *final fitness values* can be defined recursively as,

$$F(x,t) = \begin{cases} \phi(x), \text{ if } t = 0 \\ \max_{i} [F(\delta_{i}(x), t - C_{i}(x))], \text{ if } t > 0, \end{cases}$$
(1)

where $C_i(x)$ is a cost evaluation function of operator δ_i applied to state *x*. This is the kind of dynamic programming specification used in the Scatter/Gather example, and applicable to a broad class of information foraging problems.

ALTERNATIVE INTERACTION SPACES

The dynamical models of the alternative interface designs were defined on the basis of data collected in two studies of Scatter/Gather (Pirolli & Card, 1995; Pirolli et al., 1996), by variations in task conditions, and by "what-if" specifications of system improvements.

Task Conditions

The "what-if" simulations here explored two factors affecting task conditions: (1) deadlines and (2) quality of the repository relative to given queries; that is the number of items in the repository relevant to a given query. The deadline conditions were:

From this conception of the search process, the state space grows exponentially with each additional step of interaction analyzed. If one considers all interaction paths of length *L*, with *b* alternative branches from every state, then there will be L^b states in that state space. See Bertsekas (1995) for discussion of how dynamic programming problems generally grow exponentially with the number of dimensions used to represent states (e.g., *T*, *N*, and *R* in this example).

² This section may be skipped. It is provided for the mathematically inclined reader.

- *Soft deadline* of 720 seconds, which is the mean time taken by Scatter/Gather users studied in Pirolli and Card (1995) who had no time pressure in their task specifications.
- Hard deadline of 360 seconds.
- The repository quality conditions were:
- *Sparse repository* in which there were R = 303 relevant documents among the N = 742,833 total documents for a given query. This corresponds the TREC queries in the medium range of difficulty (Pirolli et al., 1996).
- *Rich repository* in which there were R = 865 relevant documents among the N = 742,833 total documents. This corresponds the TREC queries in the easy range of difficulty(Pirolli et al., 1996).

Table 1. Empirical cost estimates from Pirolli andCard (1995)

Scanning a cluster and judging relevance	$t_{SC} = 3 \text{ s}$
Adding a cluster to gather list	$t_{gc} = 5 \text{ s}$
Scanning a document title and judging relevance	$t_S = 1 \text{ s}$
Selecting, cutting, and pasting title to record window	$t_h = 5 \text{ s}$
System time to scatter new clusters	$t_{cl} = 23 \text{ s}$
System time to display titles in a cluster	$t_d = 20 \text{ s}$

Baseline System Specifications

Table 1 contains time cost estimates for various events involving the baseline Scatter/Gather system (Pirolli & Card, 1995). Using these estimates, Table 2 presents relevant state changes and costs incurred by various user actions.

To model the effects of gathering clusters and scattering them in Table 2 we use a function D(k). It models the proportion of relevant documents contained in the best kclusters presented in the Scatter/Gather state. That is if, Ris the total number of relevant documents in all the clusters on a Scatter/Gather display, then D(k) is the proportion of R that is in the k best clusters. This function is based on the analyses of Pirolli and Card (Pirolli & Card, 1997), is described in the Appendix, and plotted in Figure 4.

Table 2. Effects and time costs of state-change operators. N is the total documents and R the number of relevant documents in the current state. Time costs are described in Table 1 and D(k) in the Appendix.

Operator	New state	Time Cost (sec)
ScanDisplayTitleswindowuntilendofdisplay	Collected relevant titles= <i>R</i>	$N t_s + R t_h$
Scan Display Titles window until deadline hit after <i>t</i> sec	Collected relevant titles = $(R t)/(N t_s + R t_h)$	t
Gather k clusters and scatter	New $N = N k/10$ New $R = D(k) R$	$10 t_{sc} + k t_{gc} + t_{cl}$



Figure 4. Proportion of relevant documents collected by gathering the *k* best clusters.

Alternative System Specifications

Two system improvements were explored:

- *Faster interaction*, in which the time cost of computing a new Scatter/Gather cluster display was cut by 1/2.
- *Improved clustering*, in which the clustering algorithm was improved so that it placed 25% more relevant documents in the best cluster (see Figure 4 and Appendix).

RESULTS

The dynamic programming results show that substantial and non-intuitive trade-offs emerge regarding the performance of system improvements across task conditions. The simulated best-case gains of relevant documents for the baseline Scatter/Gather system are presented in Table 3. Against these baseline data we can examine the effects of system improvements. Overall, the best-case gains for improved clustering simulations were, on average 23% better than the baseline system, whereas the faster interaction simulations were, on average, 18% better than baseline. However, improved clustering was not always predicted to be better than faster interaction; there were, in fact, many subtle System \times Task interactions, as I show next.

Table 3. Simulated optimal number of relevantdocuments gained in the baseline Scatter/Gather system(numbers rounded to integers).

	Deadline		Mean
Repository	Hard	Soft	-
Sparse	11	49	30
Rich	16	65	40
Mean	13	57	-

$\textbf{System} \times \textbf{Repository Effects}$

Figure 5 shows the improvements predicted for a faster interaction system and a system with improved clustering, under different repository conditions. With a repository rich with relevant information, the simulations suggest there will be no major difference between two particular improvements that were examined. However, when the repository is relatively sparse with relevant information, the simulations predict that a system with improved clustering will be superior.



Figure 5. Simulated improvements in expected number of relevant documents collected by task deadline as a function of repository condition.

$\textbf{System} \times \textbf{Deadline Effects}$

Figure 6 shows simulated improvements under different deadline conditions. The simulations suggest that improved clustering will be superior when the deadlines are soft. On the other hand, when there is a hard deadline, with less time available, a system with faster interaction time will have better pay-offs.



Figure 6. Simulated improvements in expected number of relevant documents collected by task deadline as a function of deadline condition.

Effects on Strategy

The dynamic programming simulations can also afford some exploration of the optimal user strategies for the different Scatter/Gather system configurations. The optimal strategies are the action choices made along the optimal interaction path, as determined by dynamic programming. For instance, in the simple state space of Figure 3, the optimal path can be traced by following the bottommost arrows from state to state. That path is optimal because each move goes to the highest gain (G) next state. The action choice along the portion of the optimal path depicted in Figure 3 is to choose two clusters at each stage.

The optimal user strategies may vary across the different system improvements and task conditions. Here I examine the average number of clusters the ideal user would have to chose on each Scatter/Gather display, and the amount of time the ideal user would spend scanning titles for relevant results. It turns out that the dynamic programming analyses shows strategy shifts that are consistent with models developed in Information Foraging Theory (Pirolli & Card, 1997).

Table 4 shows the average number of clusters chosen from Scatter/Gather displays in the simulation of an optimal user on the baseline system. The simulations for the faster interaction system and the improved clustering system only showed differences from Table 4 in the soft deadline conditions. Under soft deadlines, the faster interaction simulations showed the same or more clusters being chosen than baseline, whereas the improved clustering simulations showed less clusters being chosen.

These results are consistent with Information Foraging Theory. The Information Diet Model (Pirolli & Card, 1997) predicts that fewer clusters should be chosen with increases in profitability of clusters (the ratio of expected relevant documents to expected processing time). This would predict the above findings of fewer clusters chosen in Rich Repository conditions and fewer clusters with improved clustering. Other predictions made by the Information Diet Model have been corroborated by empirical analyses of Scatter/Gather (Pirolli & Card, 1997), so our confidence in the dynamic programming analysis is somewhat bolstered by its agreement with these other theoretical and empirical results.

 Table 4. Average number of clusters selected by an optimal user of the baseline system.

	Deadline		Mean
<u>Repository</u>	Hard	Soft	-

Sparse	1.00	2.17	1.59
Rich	1.00	1.80	1.40
Mean	1.00	1.99	

Table 5 shows the time spent scanning the Display Titles window by an ideal user on the baseline system. Figures 7 and 8 and show the reduction in these scanning times expected for the two system improvements across the task conditions. Under optimal use, a faster interaction system would require the least scanning time in sparse repositories or hard deadline conditions, whereas an improved clustering systems would require least scanning time in rich repositories or under soft deadline conditions.

Table 5. Time spent scanning the Display Titles window by an optimal user in the baseline condition (sec).

	Deadline		Mean
Repository	Hard	Soft	
Sparse	232.0	383.0	307.5
Rich	232.0	310.0	271.0
Mean	232.0	346.5	

Again, these results are consistent with the predictions of Information Foraging Theory (Pirolli & Card, 1997). The Information Patch Model would treat the Display Titles window as an *information patch*. That model predicts that the time spent in information patches should (a) decrease as one goes from Sparse to Rich Repository conditions, (b) decrease from baseline to improved clustering systems, and (c) decrease from baseline to faster interaction systems. Again, other predictions of the Information Patch Model are corroborated by empirical analyses (Pirolli & Card, 1997), and this provides another set of consistency checks on the dynamic programming model.



Figure 7. Simulated reductions in Display Titles scanning times as a function of repository conditions.



Figure 8. Simulated reductions in Display Titles scanning times as a function of deadline conditions.

GENERAL DISCUSSION

Dynamic programming was used to explore some tradeoffs in a browser design. Specifically, the analysis explored making the browser system faster (faster interaction) and making the relevant information easier to find (improved clustering). These improvements directly address the two most common problems reported by WWW users (Pitkow & Kehoe, 1996). Dynamic programming analysis permitted the exploration of "what-if" scenarios testing these hypothetical design improvements against variations in task conditions involving repository quality and deadline conditions. Finally, the dynamic programming analysis permitted the exploration of changes in ideal user strategies across system and task conditions.

The main aim of this paper was to describe and illustrate the dynamic programming technique applied to an information foraging technology. The illustration showed how—even for this relatively simple space of designs and usage conditions—that complex and non-intuitive tradeoffs emerge from the analysis. This sort of complexity faces virtually every interface designer. Tools such dynamic programming are needed to explore and manage such design complexity.

Without doubt, the extension of this analysis to other browsers-for the World Wide Web, for instance-will not be simple. The aim here was to illustrate the technique using a tractable example, as a beachhead for more complex analyses. The analysis here assumed a very simple assessment of value: the number of relevant documents found while searching. As discussed elsewhere (Pirolli & Card, 1997; Pirolli & Card, 1995), characterizing the value of information is usually more complex, since it typically varies (at least) with tasks, In addition, individual needs, and time. the unidimensional value assessment here is overly simplistic because real-world tasks often require the assessment of information along many dimensions. Similarly, costs often involve more than just time (e.g., money). The analysis here also assumed a very simple range of user strategies and actions. This reflects the Scatter/Gather browser as used in our experimental tasks, but other tasks and browsers will undoubtedly have richer interaction spaces. None of these extensions are problematic in principle, although they may require effort to achieve in practice.

For the researcher aiming at scientific understanding of the principles underlying human-computer interaction with information systems, the dynamic programming technique can be viewed as a method for generating strong and complex hypotheses about interactions of designs, task conditions, and user strategies. It is unfortunate that the traditional study of information technology has been dominated since the Cranfield studies of the 1960's by the notion that two, and only two, factors are important to good design: (1) *precision*, which is the proportion of relevant items in a retrieved set of items, and (2) *recall*, which is the proportion of all items in the corpus that are retrieved (Harter & Cheng, in press; vanRijsbergen, 1979).

The model presented here can be taken as a rational analysis that illustrates the myopia of such a view with respect the broader complexity and trade-offs of information foraging.

ACKNOWLEDGMENT

This research was supported in part by an Office of Naval Research grant No. N00014-96-C-0097 to Peter Pirolli and Stuart Card.

REFERENCES

Bertsekas, D. P. (1995). *Dynamic programming and optimal control theory*. Belmont, MA: Athena Scientific.

Cutting, D. R., Karger, D. R., & Pedersen, J. O. (1993). Constant interaction-time Scatter/Gather browsing of very large document collections. *Proceedings of the SIGIR '93*

Cutting, D. R., Karger, D. R., Pedersen, J. O., & Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. *Proceedings of the SIGIR* '92 (pp. 318-329).

Harman, D. (1993). Overview of the first text retrieval conference. *Proceedings of the 16th Annuam International ACM/SIGIR Conference* (pp. 36-38), Pittsburgh, PA.

Harter, S. & Cheng, Y. (in press). Evaluation of information retrieval systems: A review article. *Annual Review of Information Science and Technology*.

Mangel, M. & Clark, C. W. (1988). *Dynamic modeling in behavioral ecology*. Princeton, NJ: Princeton University Press.

Pirolli, P. & Card, S. (1997). *The evolutionary ecology of information foraging* (Tech. Rep. UIR-R97-01). Palo Alto, CA: Xerox PARC.

Pirolli, P. & Card, S. K. (1995). Information foraging in information access environments. *Proceedings of the CHI* '95, *ACM Conference on Human Factors in Software* (pp. 51–58), New York.

Pirolli, P., Schank, P., Hearst, M., & Diehl, C. (1996). Scatter/Gather browsing communicates the topic structure of a very large text collection. *Proceedings of the Conference on Human Factors in Computing Systems, CHI* '96 Vancouver, BC.

Pitkow, J. E. & Kehoe, C. M. (1996). GVU's Sxth WWW User Survey. Online Publication: http://www.gvu.gatech.edu/user_surveys.

vanRijsbergen, C. J. (1979). *Information retrieval*(2nd ed.). Boston, MA: Butterworth & Co.

APPENDIX

If there are *R* relevant documents in a Scatter/Gather state, then those documents will be distributed somehow across the 10 clusters in the state. Analyses (Pirolli & Card, 1997) show that when clusters are ranked c = 1, 2, ... 10 in decreasing order by how many relevant documents they contain, then they are distributed in an exponentially decreasing fashion according to,

$$d(c) = .47 \exp(-.63(c-1)).$$
(2)

If a user collects the top ranked k clusters, then the collected proportion of relevant documents is just the sum of the proportion of relevant documents in those clusters:

$$D(k) = \sum_{c=1}^{k} d(c) \tag{3}$$

This is plotted in Figure 4. Pirolli and Card (1997; 1995) present the analysis that shows that ranking clusters and considering collections of the k = 1, 2,...K best clusters is the optimal strategy for identifying the best collection. All possible collections do not need to be explored.

For the improved clustering models we used

$$d(c) = .60 \exp(-.92(c-1)), \tag{4}$$

which is also presented in Figure 4. A detailed computational cognitive model called ACT-IF (Pirolli & Card, 1997) provides an explanation of the cognitive machinery underlying these assessments by users interacting with the Scatter/Gather interface.