Chapter 9

HyperContext Evaluation

9.1 Introduction

In this chapter the experiments to justify the HyperContext approach to adaptive hypertext are described. We conducted a series of experiments to show that the short-term user model can adequately represent a user's interests (Section 9.4), and that relevant information can be located faster with adaptive navigation support than without (Section 9.5). The final experiment consisted of users giving relevance feedback on documents recommended by Adaptive Information Discovery after traversing paths through a HyperContext hyperspace (Section 9.7).

The experiments on the short-term user model were conducted in a HyperContext hypertext created using Apple Inc.'s HyperCard, while other experiments where conducted using methods from the HyperContext prototype described in Chapter 7 and the HyperContext hypertext described in Chapter 8.

9.2 Background

Although hypertext systems have been in existence for many years, there are as yet no standard formal methods for testing them. Obtaining performance results in simulated or "live" experiments for adaptive hypertext systems is even more difficult, partly because adaptive hypertexts are a relatively new technology, and partly due to the reliance on subjective feedback from users.

Hypertext is a recent reading and information organisation method, where additional information that a user might want to access is provided via a hyperlink. In non-adaptive hypertext systems, to ensure that all users can access all the information they might need

at any point in hyperspace, it is necessary to totally connect all nodes, essentially linking everything to everything else. However, good hypertext design requires that the users are not overloaded with choice. A node containing precisely the information the user requires may be just a couple of links away, but without additional support the user may have great difficulty locating it. Adaptive hypertext systems attempt to provide a solution to this problem by anticipating what the user might find relevant, and by subverting links which might otherwise distract or confuse the user, or which would lead to non-relevant information. General-purpose adaptive hypertext systems (as opposed to Intelligent Tutoring Systems which use hypertext as their underlying model[1]) usually require the services of an information retrieval system to locate relevant information, and a user model to represent the user's interests. Both the fields of user modelling and information retrieval tend to suffer from problems with accuracy as the representation of information tends away from being based on domain knowledge. Conversely, the more domain knowledge is included within an information retrieval or user modelling system, the more expensive they become. The HyperContext framework attempts to provide a reasonably cheap method of determining and representing user interests and information without relying on domain knowledge, through the use of separate subjectively created interpretations of information in context. An interpretation represents terms which are relevant to the interpretation of the document in context, eliminating, or reducing, the ambiguity of relevance of terms which occurs in approaches similar to those taken by Ruthven and van Rijsbergen [76].

9.3 Setting the boundaries

The HyperContext hypertext, as explained in Chapter 8.7, has been constructed automatically rather than being built, over time, by a community of users. Although the approach to the creation of interpretations of information is consistent throughout the hypertext, the obvious benefits of the intelligent selection of terms to describe an interpretation is missing. However, for the purposes of the experimentation, we must accept that the interpretations we have generated are accurate, and that a user who is dissatisfied with an interpretation could, in an extended implementation of the framework, modify the interpretation or create an new one.

---

[1] See the discussion in Chapter 3.2.

9.4 The short-term user model

9.4.1 Creating a test-bed

We show that based on a path of traversal through interpretations of documents we can use the modified Rocchio method of formula 5.4 to derive a salient interpretation, and then combine the salient interpretations of a context session to derive a description of a potentially relevant document. In this experiment, a document is judged to be relevant if it is sufficiently similar to a query obtained from the user model (using the cosine similarity measure), and if the user has not already encountered the document (regardless of context) during the context session.

For this series of experiments, we decided to construct a new HyperContext hypertext, which is described below, rather than use the HyperContext hypertext described in Chapter 8, as we required a simpler environment within which to conduct the experiments. In Sections 9.5 and 9.7, we take the solutions derived from these experiments and apply them to the HyperContext hypertext of Chapter 8.

The experiments described in this section were conducted using a HyperContext hypertext created with Apple Inc.'s HyperCard, running on an Apple PowerBook 5300cs. Given the criteria we have for relevance, we decided to automate the experiment, rather than using test subjects. The over-riding reason is that we require that the results obtained are repeatable.

A hypertext of interpreted documents was automatically constructed from representations of letters of the English alphabet, using a vocabulary of 15 terms (character features) and five discrete values per feature. Figure 9.1 depicts the representations of the characters "A" and "E". Each letter is divided into 15 regions, and each region is filled with one of white space, "/", "\", "|", or "-", whichever would ultimately result in a reasonable overall character resemblance.

Once all 26 letters had been described in this way, a numeric string representation of each letter was obtained by replacing a symbol with its numeric equivalent, from the top-left to the bottom-right feature. White space is replaced by 0, "|" by 1, "/" by 2, "-" by 3, and "\" by 4. The numeric string representing the character "A" in figure 9.1a is "233341333110001". The numeric string derived in this stage for each letter forms the interpretation of the character in the context bottom.

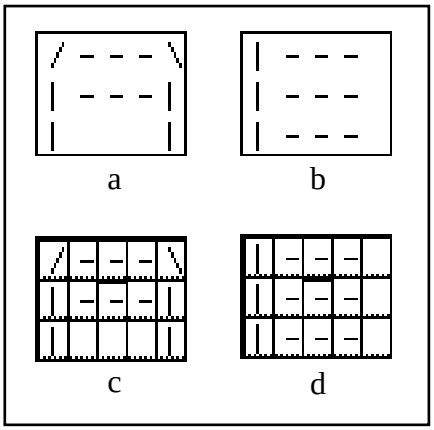

Figure 9.1: The characters "A" (a) and "E" (b) shown divided into 15 regions (c), (d)

A HyperContext hypertext was automatically constructed from 610 unique one-, two- and three-letter words, by creating nodes and links according to the following algorithm:

```
for each word
    for each letter in the word
        if the letter is the first character of the word then
            if a node does not exist for the letter then
                create a new node with the interpretation of the letter in the context
bottom
            end if
        else (if the letter is not the first character of the word)
            if a node does not exist for the letter in context of the previous letter then
                create a new node with an interpretation of the letter
            end if
            create a link from the node representing the previous letter in context to the
            node representing the current letter
        end if
    end (for each letter in the word)
end (for each word)
```

An interpretation of a letter in context is created by replacing randomly selected features in the numeric string representing the letter in the context bottom with zero (white space). For example, an interpretation of the letter "A" in some context other than bottom could be "203040303010000".

Figure 9.2 shows two nodes. On the left, the letter "A" is interpreted in the context of bottom (bottom is represented by "-"). On the right, the letter "C" is interpreted in the context of "A". "A" in the context of bottom represents "A" whenever it is the first

character of a word. "C" in the context of "A" represents "C" in each word where "A" is the preceding character. The rendition of the letter is a graphic representation of the numeric string representing the interpretation of the letter in context. The links are the letters which follow the current letter in context, so "C" in the context of "A" is only ever followed by the letters "E" or "T" (from the words "ace" and "act"). In all, there are 282 nodes in the hypertext.

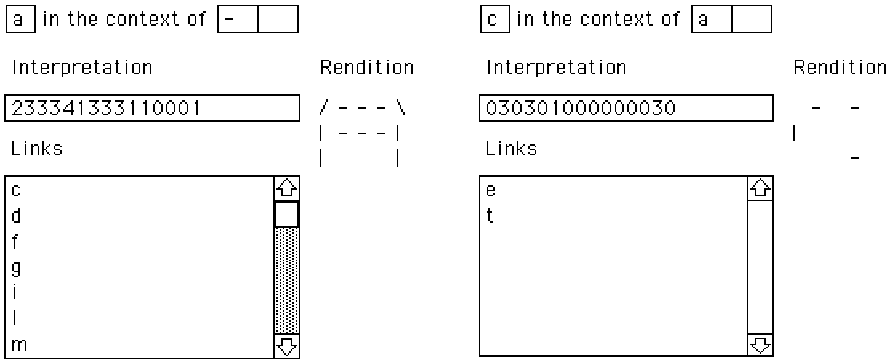

Figure 9.2: Example nodes and links to letters interpreted in context

One difference between this test HyperContext hypertext and the HyperContext framework, is that in the framework a context is represented by a node-label pair. In this experimental hypertext, we have dropped label from the context. If we were to represent label, we could use one of the 15 character features (see figure 9.1). However, we are already able to partition the hyperspace using node only to represent the context (figure 9.3), which is sufficient for the purposes of the experiment. Using label as well would serve to increase the number of interpretations. To comply with the HyperContext framework, we can say that the name of a link's destination letter is the name of the link. In figure 9.2 we can say that the interpretation of "C" in the context of "A" is really the interpretation of "C" in the context of node "A" and label "C", as label names can be arbitrary as long as they uniquely identify a link.

This hypertext is used to determine whether a user model derived from paths of traversal can be used to identify a letter which has not already been encountered on the path of traversal, and which is sufficiently similar to a query derived from the user model to be considered relevant. This implies that an information retrieval method is used to determine the degree of relevance. We use the cosine similarity measure (formula 9.1) to determine degree of relevance.

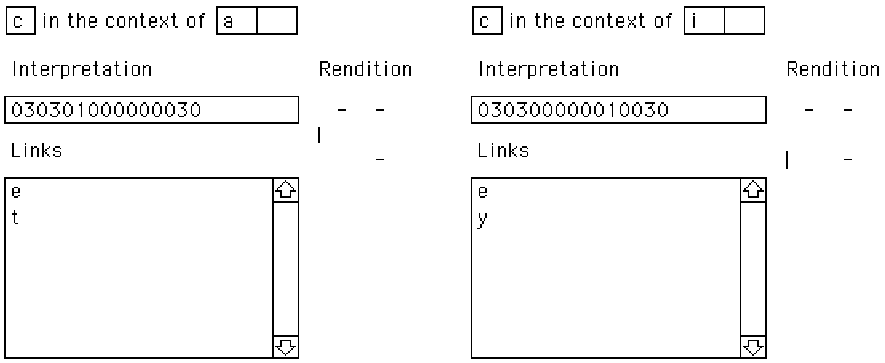

Figure 9.3: "t" is accessible from "c" only when "c" is accessed in the context of "a"

In formula 9.1, t ranges over a vocabulary of size N of terms, q_t is the weight of the tth term in the query, and d_t is the weight of the tth term in the document being evaluated. The document with the highest $\cos(\theta)$ is the document most similar, and consequently most relevant, to the query.

$$\mathbf{cos(\theta)} = \frac{\sum_{t=1}^{N} q_t \times d_t}{\sqrt{\sum_{t=1}^{N} q_t^2 \times \sum_{t=1}^{N} d_t^2}} \qquad \text{Formula 9.1}$$

The weight of a term in a document is normally calculated using the TFxIDF metric referred to in Chapter 5.2. The Term Frequency is multiplied by the Inverse Document Frequency to obtain the degree of importance of the term to the document, taking into account the overall distribution of the term throughout the document collection. In this experiment, each letter is represented by 15 features, each of which may contain one of the five values " ", "/", "-", "|", and "\". We analyse the interpretations of the letters in the context bottom to determine how good a feature with a particular value is at identifying a letter. The more frequently a feature has the same value in the representation of different letters the less discriminatory the feature-value pair is, and the less likely we are to correctly identify a single letter based on that pair alone. Figure 9.4 shows the weighted values for "-" for each feature. The higher the value of a feature, the better the chances of uniquely identifying a letter with a "-" for that feature, because fewer letters share that feature.

Each value is obtained by dividing 26 (the number of letters in the English alphabet) by the number of times the corresponding feature value occurs in a letter. For instance, feature 15 in figure 9.4 has the value 26. A "-" occurs in feature 15 of all letters only once, in the letter "I". On the other hand, feature 3 in figure 9.4 has the value 1.625, as 16 letters have the value "-" in feature 3. Consequently, given no other information, the

presence of "-" in feature 15 uniquely identifies the letter it belongs to as the letter "I", but a "-" in feature 3 is a poor discriminator.

	Weights for 3 (-)
1	6.5
2	1.625
3	1.625
4	1.857143
5	6.5
6	26
7	3.25
8	2.888889
9	2.888889
10	26
11	6.5
12	2.166667
13	1.857143
14	2.6
15	26

Figure 9.4: The weights for "-" (value 3 in the numeric string representation)

9.4.2 Automatically generating paths of traversal

A path is traversed by randomly selecting a start node of a letter in the context bottom and randomly selecting a link to another letter in the context of the start node. An available link from that interpreted letter to another letter is randomly selected. This process continues until an interpreted letter with no links is reached. A total of 599 paths with an average path length of 4.52 were automatically generated.

9.4.3 Description of the experiments

In order to determine the most consistent approach to deriving the user model based on paths of traversal, we tested three hypotheses, each performed in two categories. To construct a model of a short-term interest we first obtain a salient interpretation of an accessed interpretation before combining it with the salient interpretations of previously accessed interpretations from the same context session.

The three hypotheses for the derivation of the salient interpretation are called interpretation+, interpretation-, and plain (which are described below). In Chapter

5.8.2 we describe a weighted scale of confidence which is used to weight a salient interpretation's ability to predict a relevant document. The nearer to the start of the context session the less confidence we have in the salient interpretation's ability to predict a relevant document. The later a salient interpretation occurs in a context session, the greater the confidence we have in it. Consequently, each hypothesis is also tested with weighted and unweighted salient interpretations.

Finally, we use control to provide a benchmark against which we can compare the performance of the three hypotheses for predicting a relevant document.

interpretation+

This hypothesis predicts that a salient interpretation which represents an interest in the document can be derived by using the features of the accessed interpretation of a document which occur infrequently in other interpretations of the same document. These *pertinent features* of the accessed interpretation are extracted using formula 5.2, which is repeated here as formula 9.2 (see Chapter 5.8.1 for an explanation of the terms).

$$I_{salient} = \alpha I_{sel} - \beta I_{ave} \qquad \text{Formula 9.2}$$

interpretation-

On the other hand, interpretation- predicts that the pertinent features of the accessed interpretation actually distract attention from what the user is likely to be interested in, because if the interest is in the pertinent features of the accessed interpretation, then the context session would probably not be extended. Consequently, we use features of the average interpretation for this document as an indicator of what the interest might be, reducing the effect of the accessed interpretation's pertinent features (formula 5.4, repeated as formula 9.3).

$$I_{salient} = \alpha I_{ave} - \beta I_{sel} \qquad \text{Formula 9.3}$$

plain

plain predicts that the user is interested in precisely the description of the accessed interpretation, regardless of the distribution of terms in other interpretations of the same document. In plain, $I_{salient} = \beta I_{sel}$.

For the interpretation+, interpretation- and plain experiments the value of α and β is 1.

control

In the control experiments, the salient interpretation of each letter in a path is represented by its interpretation in the context bottom. We consider this to be an adequate control experiment because in typical Information Retrieval Systems with relevance feedback, there is only one representation, or interpretation, available for each document.

9.4.4 Generating a query from the user model

The HyperContext framework provides for a query to be generated from the user model according to the requirements of the external information retrieval (IR) system through an interface. The query terms are presented to the IR interface as a vector of weighted terms. In this experiment, we use a vector-based information retrieval system built around the cosine similarity measure given in formula 9.1. A query takes the form of a 15 term vector obtained from the 15-digit numeric string representing the user model which is then compared to the 15-digit representations of each letter of the alphabet in the context bottom[2]. The letter which has the highest similarity to the query is considered relevant. Occasionally, either no letter is sufficiently similar to the query, or else more than one letter has the same degree of similarity to the query.

9.4.5 Results

Figures 9.5 and 9.6 show the numbers of predictions as raw data and in percentage terms respectively. For each of the values for interpretation+, interpretation-, plain, and control, the left-hand column (+) represents results obtained using the weighted confidence scale, and the right-hand column (-) represents results obtained without using the confidence scale.

No Prediction is the number of times the experiment failed to predict any letter at all. The Letter in Word and Letter Last are the number of times the predicted letter had already been encountered in the path of traversal, and of these, the number of times the predicted letter was the same as the last letter encountered in the path. Finally, results are given for the number of times a previously unseen letter was predicted.

---

[2] In a normal HyperContext environment, the most relevant interpretation in any context would be identified.

Plain:				Interpretation+:			
No Prediction+	15	2	-	No Prediction+	358	319	-
Letter in Word+	302	344	-	Letter in Word+	121	138	-
Letter Last+	136	92	-	Letter Last+	32	27	-
Unseen Letter+	282	253	-	Unseen Letter+	120	142	-

Control:				Interpretation-:			
No Prediction+	0	0	-	No Prediction+	108	158	-
Letter in Word+	477	467	-	Letter in Word+	148	135	-
Letter Last+	254	166	-	Letter Last+	88	86	-
Unseen Letter+	122	132	-	Unseen Letter+	343	306	-

Figure 9.5: Results in raw figures

The number of times an unseen letter was predicted has already been reported in Chapter 6.7.1. interpretation- predicted an unseen letter 57.3% of the time, followed by plain with 47.1%, interpretation+ (20%) and control (20.4%), when the weighted confidence scale is used. If the number of times an already encountered letter is taken into account, then we see that control overwhelmingly favours letters already encountered (79.6%), and significantly when the recommended letter is a letter already encountered in the path, it is the same as the last letter in the path 53.3% of the time. When the confidence scale is not used, although an encountered letter is recommended 78% of the time, the number of times it is the same as the last letter in the path is just 35.6%.

Plain:	%	%		Interpretation+:	%	%	
No Prediction+	2.5	0.0	-	No Prediction+	59.8	53.3	-
Letter in Word+	50.4	57.4	-	Letter in Word+	20.2	23.0	-
Letter Last+	45.0	26.7	-	Letter Last+	26.5	19.6	-
Unseen Letter+	47.1	42.2	-	Unseen Letter+	20.0	23.7	-

Control:	%	%		Interpretation-:	%	%	
No Prediction+	0.0	0.0	-	No Prediction+	18.0	26.4	-
Letter in Word+	79.6	78.0	-	Letter in Word+	24.7	22.5	-
Letter Last+	53.3	35.6	-	Letter Last+	59.5	63.7	-
Unseen Letter+	20.4	22.0	-	Unseen Letter+	57.3	51.1	-

Figure 9.6: Results in percentage terms

We also require that the number of times a letter is recommended is significantly high. control and plain virtually always make a prediction, interpretation+ fails to make a prediction 59.8% of the time, and interpretation- 18% of the time.

If the confidence scale is not used, then the number of times a predicted letter has already been encountered in the path of traversal increases in all cases except for interpretation- and control. However, in interpretation- without the confidence scale, the number of unseen predictions drops from 57.3% to 51.1%, although in control it rises slightly from 20.4% to 22%.

9.4.6 Prediction performance

In this section, we take the results of Section 9.4.5 and compare the observed frequency of predicting an unseen letter to the probability of randomly predicting an unseen letter.

In a random environment, the probability of predicting an unseen letter is 1 - *P*, where *P* is the probability of predicting a seen, or already encountered, letter. *P* is $\frac{N}{26}$, where *N* is the number of unique letters in the traversed path, and 26 is the number of letters in the alphabet. When the path consists of one letter, the probability of predicting the same letter is $\frac{1}{26}$, so the probability of predicting a different letter is 1 - $\frac{1}{26}$. When the paths consists of all 26 letters, then the probability of predicting a different letter is 0 (1 - $\frac{26}{26}$), and when there are 13 unique letters in the traversed path, the probability of predicting one of the other 13 letters is 1 - $\frac{13}{26}$, or 0.5.

The data created in Section 9.4.1 consisted of 599 paths of traversal each containing from 2 to 9 unique letters. Table 9.1 provides the probabilities of randomly predicting a letter not already encountered in the path and the actual observations from control and interpretation- using the weighted scale of confidence.

Table 9.1 shows that the Rocchio method employed in control is generally reluctant to move the centroid from the representation of the average letter of the traversed path, regardless of path length. On the other hand, the modified Rocchio method of interpretation- generally has a significantly better chance of doing so. Although interpretation- falls short of the probabilities of a purely random environment, it does have the significant advantage that it will always predict the same letter for a given path, regardless of how frequently that path is traversed. In a purely random environment, subsequent traversals of the same path would normally result in the prediction of different letters.

No. of unique letters in path of traversal *N*	% no. of paths of *N*	Probability of predicting an unseen letter 1 - (*N*/26)	Actual observations from interpretation-	Actual observations from control
2	6.0	0.92	0.64	0.19
3	40.9	0.88	0.69	0.24
4	22.7	0.85	0.50	0.20
5	13.0	0.81	0.50	0.15
6	10.7	0.77	0.48	0.16
7	4.2	0.73	0.44	0.16
8[3]	1.7	0.69	0.10	0.20
9[3]	0.8	0.65	0.20	0.40

Table 9.1: Comparison of probabilities of recommending an unseen letter

All in all, the most consistent approach to deriving the user model is that afforded by interpretation- to generate the salient interpretation, in combination with the weighted confidence scale.

9.5 Locating relevant information

9.5.1 Ideal empirical study scenario

Ideally, the HyperContext hypertext is tested using groups of users to determine whether an adaptive hypertext constructed using the HyperContext framework gives advantages over and above a non-adaptive hypertext.

Originally, an empirical study was planned using three groups of six users, after a selection process to categorise users as novice, intermediate and advanced information seekers. Each group would consist of two novice, two intermediate and two advanced information seekers. One group would form the control group, and use the non-adaptive version of HyperContext described in Chapter 7.8, to provide baseline measurements. The other two groups would use the adaptive HyperContext. Each group would have identical sets of tasks to perform, estimated to last approximately 2.5 hours in all, and performance data about the efficiency with which each task was executed would be automatically logged by measuring the number of nodes accessed, and the number and duration of deviations from the "ideal" paths from the start node to the target node.

---

[3] These values are based on too small a sample to be considered accurate.

A number of factors affected the prospects of conducting such an empirical study. While some factors were concerned with logistics, the major set-back is that some of the HyperContext prototype has not been implemented as a multi-user system, meaning that it is not possible to run concurrent user sessions. Sequential user experiments also proved impractical to organise. However, the spirit of the empirical study has been retained, and the number of tasks has been increased, but the experiments have been automated.

9.5.2 Automatic evaluation of the HyperContext hypertext

The spirit of the automated experiments is that required information can be located faster in an adaptive HyperContext hypertext than in a non-adaptive HyperContext version. Starting from a given root node, which is always the root document www.w3.org/Overview.html, we count the number of nodes that are accessed before a given target node is located. The experiment is conducted 73 times, using randomly selected, but unique, documents which are not direct children of, but which are reachable from, the root node. Each experiment has four phases. In the first phase, we see how many nodes must be accessed using a brute-force hybrid depth-first search through a non-adaptive hypertext (consisting of document interpretations in the context bottom only). In the third, we count the nodes accessed using the same hybrid search method applied to the adaptive hypertext (consisting of separate document interpretations for each context). The second and fourth phases employ a link ordering algorithm in the non-adaptive and adaptive hypertexts respectively.

For brevity, the four phases are referred to by the following acronyms: phase 1 is NABF (Non-Adaptive Brute Force); phase 2 is NAO (Non-Adaptive Ordered); phase 3 is ABF (Adaptive Brute Force); and finally, phase 4 is AO (Adaptive Ordered).

There are four possible outcomes when comparing the results of NABF and ABF:

- they require the same number of node accesses;
- ABF is faster, because some earlier links have been deleted;
- ABF is slower, because the first link to the target document in the original hypertext has been deleted, but there is another, later, path to it in the adaptive hypertext;
- ABF fails because a link to the target document has been deleted.

Ideally, the average number of nodes accessed is less in ABF than in NABF.

In NAO and AO, rather than accessing nodes using a brute-force search method based on the natural link order, we first use a link ordering algorithm to order links according to relevance. The link ordering algorithm is based on the comparison of the salient interpretations of a node's children to the target node. The nodes are then accessed in order of similarity to the target document, rather than their order of occurrence. Although link ordering by similarity is trivial when the target node is a child of the currently accessed node, its overall success or failure depends on its ability to direct search during the earlier stages of the context session. For example, consider that the root document has ten children, and using the brute-force approach to reach the target it is necessary to traverse the fifth link. If the link ordering algorithm incorrectly re-orders the links so that the required link would be traversed later rather than earlier, then more nodes will need to be accessed prior to the target being reached (when compared to the brute-force adaptive and non-adaptive approaches). Conversely, if the required link is traversed earlier, then less nodes will be accessed on the way to reaching the target.

We use a hybrid depth-first approach to the hypertext traversal. As the hypertext contains many navigational links, a pure depth-first search would result in greater search times for the target document, because several links which naturally occur earlier are navigational links enabling the browser to rapidly relocate to landmark nodes such as the root node. The hybrid approach is actually based on a breadth-first approach. The disadvantage of a pure breadth-first search is that all nodes at each level need to be checked until the level which contains the target node is reached. For example, if the target node is three levels from the root node, then all nodes at the intermediate levels need to be accessed, regardless of how well, or badly, the links have been ordered. The benefit of the link ordering algorithm would only be seen at the comparison of siblings of the target node. The hybrid depth-first search algorithm maintains a structure which represents the order in which nodes are to be accessed. Once the target node has been located, we revisit the structure to remove nodes which need not have been accessed, so that we can count the number of nodes which must be visited to reach the target, based on the link order. Consider the structures in figure 9.7.

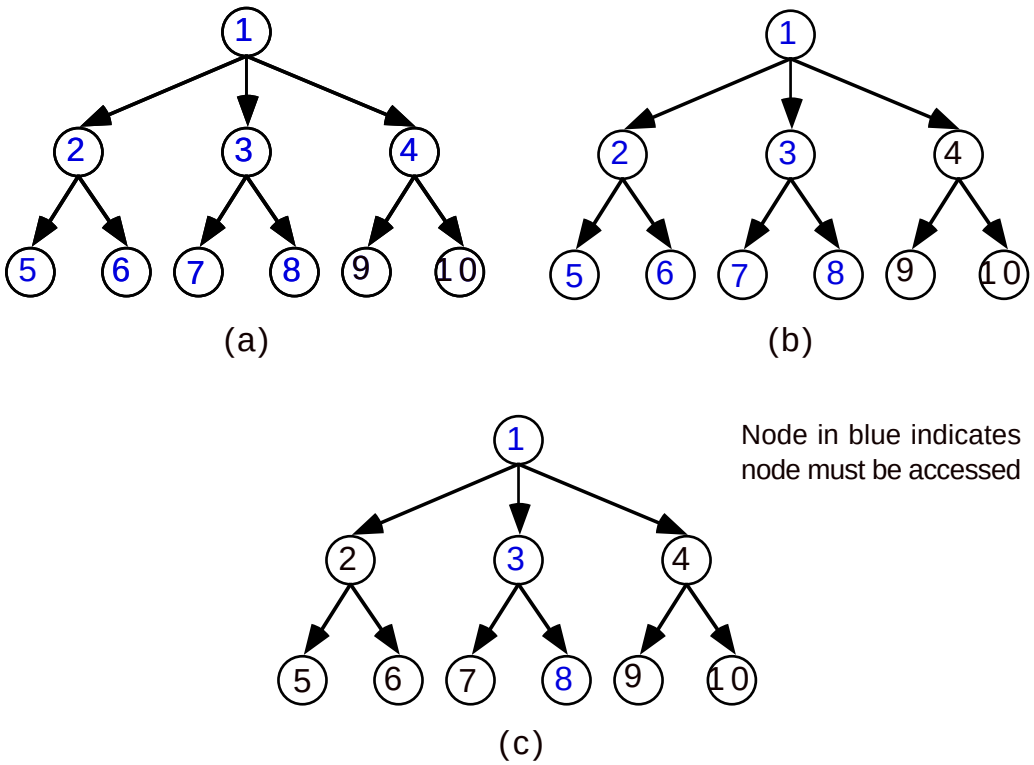

Figure 9.7: Locating a target node (a) target node found by brute-force breadth-first search. (b) represents the nodes which need to be accessed with prior knowledge of the level depth of the target node. (c) represents the nodes to access with optimal link ordering.

Figure 9.7a represents the nodes which must be accessed using a brute-force breadth-first method. Figure 9.7b represents the nodes which would have been accessed had an *n* look-ahead depth-first strategy been used. *n* is a "magic" number which represents the level depth of the target node from the root node. For example, if *n* = 0, there is no look-ahead (the root node is the target node); if *n* = 1 we look-ahead to the root node's children; and if *n* = 2 we look-ahead to the root node's grandchildren. In figure 9.7b, if we know in advance that the target document is a grandchild of the root node, using the magic look-ahead level, there would be no need to access node 4. The hybrid depth-first search strategy builds a tree structure representing the nodes that are accessed using a breadth-first search until the target node is located (for example, the blue coloured nodes in figure 9.7a). The shortest path in the tree to the target node from the root is identified (represented in figure 9.7c). The tree is then revisited to remove all those nodes which exist on the right-hand side of the shortest path (figure 9.7b). The remaining nodes are those which must be accessed by the hybrid depth-first search. Figure 9.7c also shows the optimal ordering of the links by the link ordering algorithm. The level depth represents the minimum number of nodes which must be accessed to reach the target node from the root using the shortest path.

Another difference between a pure depth-first approach and the hybrid approach we have adopted is depicted in figure 9.8. The hybrid approach will locate the shortest path between the root node and the target and count the number of nodes to the left of this path (including the nodes in the path), rather than the minimum number of intervening nodes. In figure 9.8, the target node, node 8, is linked to from nodes 3 and 5, at level 2 and level 3 respectively. As the hybrid depth-first approach is really a modified breadth-first approach, the search will terminate when node 8 linked to from the node 3 is located. A pure depth-first approach would have first located the target node at level 3 (via node 5) which would have been found after accessing less intervening nodes than the target node at level 2. HyperContext is able to direct users along the path which requires the least number of intervening nodes (that is, to the target node at level 2), or to the target node which is located first by a parallel search algorithm in Information Retrieval-in-Context and Adaptive Information Discovery. For the purposes of these experiments, we are less interested in algorithmic efficiency than we are in effective link ordering. We are more interested in demonstrating that the link ordering algorithm using interpretations of information is more useful than a link ordering algorithm which does not distinguish between descriptions of the same information in different contexts.

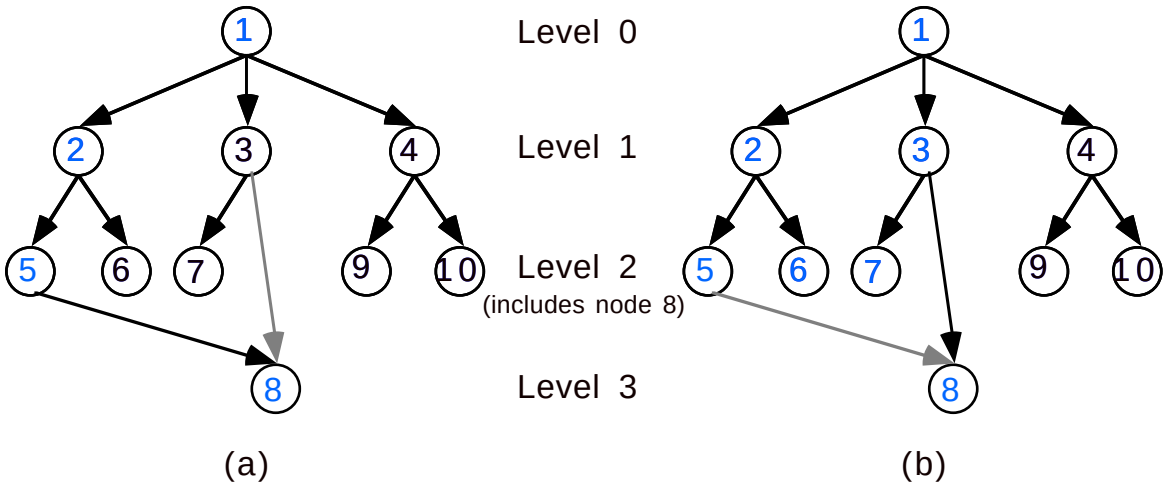

Figure 9.8: Difference between pure and hybrid depth-first searches. Pure depth-first search (a) will locate node 8 more efficiently than hybrid depth-first (b)

The NAO and AO phases of each experiment are conducted on the adaptive and non-adaptive versions of HyperContext using a link ordering algorithm. NAO acts as the control. The brute-force phases on both the non-adaptive and adaptive hypertexts (NABF and ABF) act as an upper bound on the corresponding link ordering algorithm. Ideally, the link ordering experiments should, on average, out-perform the brute-force phases. The target node's level depth acts as a lower bound. The performance of experiments on

the adaptive hypertext should also tend to out-perform the performance of the corresponding experiments on the non-adaptive hypertext.

The experiments on the adaptive hypertext can fail to locate a target node because a node which is reachable from the root node in the non-adaptive hypertext may become unreachable as a result of the conversion to an adaptive hypertext. We have chosen to exclude unreachable nodes and nodes which are immediately accessible from the root node from the performance results of the experiments. Inclusion of these nodes would merely serve to skew the results. Therefore, of the 170 nodes in the hypertext, 70 are not tested because they are children of the root node, and a further 27 are not tested because they are unreachable from the root node in the adaptive hypertext.

9.5.3 Description of the experiments

An experiment consists of four requests to locate a given target document in context from some start document in context. The start document is always the document www.w3.org/Overview.html in the context bottom.

Each experiment has four phases: non-adaptive brute-force (NABF), non-adaptive ordered (NAO), adaptive brute-force (ABF), and adaptive ordered (AO). We are particularly interested in the results of AO when compared to NAO. The brute-force phases provide upper bounds for the corresponding ordered phases.

The non-adaptive phases (NABF and NAO) always use the interpretation of a document in the context bottom to act as the salient interpretation of the document, and to obtain the interpreted document's out-links. On the other hand, the adaptive phases (ABF and AO) use the actual description of the interpreted document in context, and also obtain the document's out-links from its interpretation. In the adaptive phases, the salient interpretation of a accessed document is obtained using interpretation- (see Section 9.4.2 and Chapter 5.8.1).

The results of the experiments are obtained and evaluated in two stages. First an automatic node locator is invoked, with parameters to identify the target interpretation and to indicate which phase of the experiment to run. The locator produces output which identifies whether target was located, the level depth at which it was located, the number of nodes which were visited (using pure breadth-first search), and the number of nodes visited using the hybrid depth-first approach. It also produces some other output useful for debugging purposes, but irrelevant for the performance comparisons. The second

stage is the automatic results evaluator, which compares the data produced by the locator and which evaluates the overall performance of each phase of each experiment.

9.5.4 Results

These results are based on the evaluation of an automatically generated HyperContext hypertext, without any attempt to improve the descriptions of interpretations and quality of links.

The first result, given in table 9.2, reports the number of experiments in which ABF outperformed NABF.

Phase	No. of successes
Adaptive brute-force:	49
Non-adaptive brute-force:	18
Same:	6

Table 9.2: Comparison of results of ABF and NABF

In ABF and NABF, links are accessed in the order in which they appear in the document. The main reason for the improvement is due to the removal of links from the adaptive version of the hypertext to nodes which are considered to be irrelevant.

Phase	No. of successes
Non-adaptive ordered:	64
Non-adaptive brute-force:	8
Same:	1

Table 9.3: Comparison of results of NAO and NABF

Tables 9.3 and 9.4 compare NAO with NABF and AO with ABF, to demonstrate the effectiveness of the link ordering algorithm on the non-adaptive version of the HyperContext hypertext and the adaptive version, respectively.

Phase	No. of successes
Adaptive ordered:	32
Adaptive brute-force:	40
Same:	1

Table 9.4: Comparison of results of AO and ABF

Table 9.3 shows a dramatic improvement in node locatability in the same non-adaptive hypertext when link ordering is utilised, based on the similarity between each of a node's link destinations and the target node.

Whereas in table 9.3 the link ordering algorithm outperforms brute-force more often than not (in the non-adaptive hypertext), in table 9.4 the adaptive brute-force method often outperforms the link ordering algorithm in the adaptive hypertext. The link ordering algorithm is advantageous 43.8% of the time. 54.7% of the time, the brute-force method is more advantageous as the link ordering algorithm incorrectly demoted the link leading to the target document. However, we must remember that these results are obtained from an automatically constructed adaptive hypertext. HyperContext is a social hypertext which can be modified to reflect usage. Such changes are highly unlikely to result in any improvements to results obtained by the non-adaptive version of the hypertext, but they can have a dramatic effect on the link ordering algorithm for the adaptive version of the hypertext. Interpretations can be modified, and new links in context can be created, both of which can positively effect the link ordering and brute-force algorithms in the adaptive hypertext. However, unless nodes' descriptions in the context bottom are modified, these modifications will have no impact on the respective algorithms' performance in the non-adaptive hypertext.

Phase	No. of successes
Adaptive ordered:	32
Non-adaptive ordered:	39
Same:	2

Table 9.5: Comparison of results of AO and NAO

Similarly, the link ordering algorithm in the non-adaptive hypertext performs better than the same algorithm in the adaptive hypertext (table 9.5).

In table 9.2, we compared the results of adaptive brute-force with non-adaptive brute-force. The former phase outperformed the latter in a significant number of experiments. In table 9.5 we compare the results of the link ordering algorithm in the non-adaptive and adaptive hypertexts, and we see that the balance is slightly in favour of the non-adaptive ordered phase of the experiments. Given that in tables 9.4 and 9.2 we saw that the adaptive brute-force phase also tended to perform better than the adaptive ordering and non-adaptive brute-force phases, the temptation might be to consider that the adaptive brute-force approach provides enough of an advantage over the link ordering algorithm to discount the link ordering algorithm as an effective method of recommending links. The adaptive brute-force approach alone demonstrates that using multiple interpretations of

information can provide an advantage by automatically partitioning the hyperspace into relevant and non-relevant interpretations of nodes. In normal HyperContext usage, link ordering is not directly used to perform link recommendation. However, link ordering does demonstrate that when retrieval is based on *interpretations* of information, there is a general increase in performance. Furthermore, in 43.8% of the experiments in which adaptive ordering is used (compared to adaptive brute-force and non-adaptive ordered) there is an improvement. This provides additional evidence that using multiple interpretations and deriving the salient interpretation using interpretation- is a valid approach.

Tables 9.6 and 9.7 are provided for completeness. Table 9.6 shows that adaptive ordering is more advantageous than the non-adaptive brute-force approach, as we have come to expect.

Phase	No. of successes
Adaptive ordered:	55
Non-adaptive brute-force:	18
Same:	0

Table 9.6: Comparison of results of AO and NABF

Table 9.7 confirms that the adaptive brute-force phase continues to outperform any ordered phase, although the link ordering algorithm continues to perform well in more than 40% of the experiments.

Phase	No. of successes
Adaptive brute-force:	41
Non-adaptive ordered:	30
Same:	2

Table 9.7: Comparison of results of ABF and NAO

9.5.5 Breakdown of the results according to path length

It is important to remember that the automatic conversion of the non-adaptive hypertext into an adaptive hypertext has had an impact on the physical dimensions of the hypertext, and the organisation and connectivity of nodes within it. This in turn means that a node located at one level in the non-adaptive hypertext can occur at a different level in the adaptive hypertext. The conversion process does not create new links, although it can remove links. For example, in figure 9.9a, node d is reachable from nodes b and c.

During the conversion process, node d is considered irrelevant when reached from node b, so the link is removed. In figure 9.9b, node d is still reachable from node a (the root node), but whereas in the non-adaptive hypertext, node d first occurs at level 2, in the adaptive hypertext, node d first occurs at level 3. In an experiment to locate node d from node a, NABF and NAO will locate node d at level 2, while ABF and AO will locate the node at level 3. In this section we show the aptitude of each phase of the experiments at locating the target node at different level depths in the hypertext. In tables 9.8 to 9.12 inclusive, the reason that NABF and NAO fail to register any successes at level depth 4 is due to all nodes being reachable by level 3 in the non-adaptive hypertext. However, in the cases were the adaptive phases register successes at level 4, this still means that the non-adaptive phases took longer than the adaptive phases to locate the target node even though in the non-adaptive hypertext the target node exists at level 3 or higher.

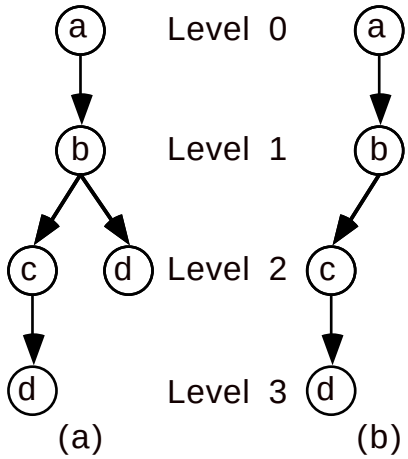

Figure 9.9: Effect of link deletion during the conversion process. Node d will first be located at level 2 in the non-adaptive hypertext (a), but at level 3 in the non-adaptive hypertext (b)

This stage of the evaluation of the experiments consists of separating the results according to the level depth of the target node, to see the overall performance of each phase at each level and when each phase is successful (table 9.8).

Phase \ Level	2	3	4	Total
non-adaptive brute-force	3	0	0	3
non-adaptive ordered	19	9	0	28
adaptive brute-force	0	15	9	24
adaptive ordered	2	12	1	15
Total	24	36	10	70

Table 9.8: Number of successes by phase by level depth of target node

The non-adaptive brute-force phase proved the best algorithm in just 3 of 70 experiments. All of NABF's successes occur when the target node is 2 levels from the root (i.e., one of the root's grandchildren). The phase with the overwhelming success at locating a target node when it is a grandchild of the root node is the non-adaptive ordered phase, with 19 of 24 successes at this level. Furthermore, this phase of the experiment is the most successful overall, with a total of 28 successes in 70 experiments. However, most of its successes occur when the target node is close to the root node. As the target node increases in distance from the root node, the adaptive algorithms register their successes. When the target node is located at level 3, the combined adaptive solutions register 27 successes, compared to the combined non-adaptive 9 successes. At level 4, the combined adaptive solutions are successful in all 10 experiments. In the remaining 3 experiments, successes were tied between two phases, in each case, and so they have not been included in table 9.8. A summary of the results of these three experiments is provided in table 9.9, although no conclusions are drawn from them, except for noting that the trends identified in table 9.8 are generally followed.

Phase \ Level	2	3	4	Total
non-adaptive brute-force	0	0	0	0
non-adaptive ordered	1	1	0	2
adaptive brute-force	0	0	2	2
adaptive ordered	1	0	1	2
Total	2	1	3	6

Table 9.9: Summary of the three "invalid" experiments

The overall sample size is not large (70 "valid" experiments[4], out of only 73 experiments), and the path length is short (all of the target nodes were first located at most 3 levels from the root in the non-adaptive hypertext, and 5 levels from the root in the adaptive hypertext), so generalising the results may be both presumptuous and premature. However, if we accept that we must be cautious when interpreting the results, an apparent trend in the experiments is that as the path length *increases* the adaptive solutions provide better results. We can re-interpret the results presented in Section 9.5.4, which indicated that the non-adaptive ordered phase (NAO) of the experiments was the most successful when compared to the general performance of the other phases. From table 9.8, we can see that although NAO has the highest number of individual successes, the combined adaptive phases just outperform the combined non-adaptive phases, and the adaptive phases, both combined and individually, outperform their non-adaptive counterparts at all levels other than when the target node is a grandchild of the root node.

[4] A "valid" experiment is an experiment which has just one successful phase.

9.5.6 Comparison of number of nodes visited by the different phases

In this section we report the average number of nodes that are visited in each phase using the pure breadth-first and hybrid depth-first search strategies, to demonstrate the effectiveness of the hybrid depth-first approach. Table 9.10 shows these results for each phase across all 73 experiments regardless of whether the phase was the successful phase.

From table 9.10 we see that NAO visited the least number of nodes, when using both the breadth-first and hybrid depth-first approaches. ABF and AO were, on average, each within 2 nodes of NAO using the breadth-first approach. This margin increased, on average, in both phases using the hybrid depth-first approach, with ABF visiting nearly 4 nodes less than AO on average.

	Nodes Visited	
Phase	Breadth-First	Depth-First
NABF	117.22	45.83
NAO	99.42	29.88
ABF	101.38	32.88
AO	99.97	36.42

Table 9.10: Average number of nodes visited overall by phase

NABF visited a significantly larger number of nodes than any other phase in either approach.

Level	2		3		4		5	
Phase	B	D	B	D	B	D	B	D
NABF	81.40	17.72	138.85	60.48	0	0	0	0
NAO	71.36	7.36	114.04	41.60	0	0	0	0
ABF	75.50	20.59	120.00	33.09	127.83	43.33	138.50	70.50
AO	73.77	29.68	100.42	34.29	126.89	45.89	130.00	59.00

Table 9.11: Average number of nodes visited in each phase, broken down by the level at which target node is located, using both the pure breadth-first (B) and hybrid depth-first (D) approaches

In table 9.11, the results of table 9.10 are broken down into the number of nodes visited at each level. Here we see that the adaptive phases visit more nodes using the hybrid depth-first (D) approach to reach target nodes at level 2, than the non-adaptive phases. This trend is reversed at level 3. Comparative results are unavailable for levels 4 and 5,

because in the non-adaptive phases all nodes are reachable by level 3. However, the hybrid depth-first results for the adaptive phases at levels 4 and 5 compare favourably with the hybrid depth-first results for the non-adaptive phases at level 3. Also, as the path length increases, the AO results improve consistently when compared to the average number of nodes visited by ABF. However, the number of target nodes located at level 5 (two nodes) is too small to predict that this trend would continue as the path length increases beyond five levels. The number of nodes located at each level for the adaptive and non-adaptive phases is provided in table 9.12.

Hypertext \ Level	2	3	4	5
Non-Adaptive	25	48	0	0
Adaptive	22	31	18	2

Table 9.12: Number of nodes at each level by hypertext

In table 9.13, we again provide the average number of nodes visited by each phase broken down by level depth, except that we count nodes only for successful phases in each experiment (consequently there are no results at level 5, as all target nodes are located by level 3 in the non-adaptive phases, and by level 4 in the adaptive phases). The actual number of successes by each phase at each level is presented in table 9.8.

Level	2		3		4	
Phase	B	D	B	D	B	D
NABF	82.00	16.33	0	0	0	0
NAO	67.26	4.11	116.67	38.44	0	0
ABF	0	0	99.73	30.80	131.11	36.11
AO	67.50	6.50	92.17	25.33	123.00	35.00

Table 9.13: Average number of nodes visited during successful phase, broken down by level at which target node is located. The average number of nodes is presented for both the pure breadth-first (B) and hybrid depth-first (D) approaches

The final data we present in this section compares the performance of each phase, broken down by level, for when the target node located by each phase is found at the same level. This table should be compared with table 9.8. In 20 of the 70 valid experiments, the target nodes were located at different levels by the adaptive and non-adaptive phases.

In table 9.8, the inclusion of the twenty experiments which resulted in the target node being located at different levels in the different hypertexts suggested that as the path length increases, the adaptive phases register an advantage over the non-adaptive phases.

In table 9.14 we remove these experiments from the results to see their effect. Once again, the sample size and the path lengths involved are too small to provide a safe basis from which to generalise. However, when the target node is located at level 2, the non-adaptive phases are clearly more successful than the adaptive phases. At level 3, however, the adaptive phases are more successful than the non-adaptive phases.

Phase \ Level	2	3	Total
NABF	3	0	3
NAO	16	2	18
ABF	0	15	15
AO	2	12	14
Total	21	29	50

Table 9.14: Successes by phase when target is at the same level

9.6 Observations

In Section 9.4.5 we presented the results of experiments with several hypotheses to establish which is more suitable for establishing a salient interpretation of a user's interests in the current document in context, and how to combine the salient interpretations in a context session to represent a user's short-term interests. In Sections 9.5.4 and 9.5.5 we present the results of experiments to determine whether multiple interpretations of information can adequately partition a hyperspace to assist with adaptive navigation to lead the user to relevant information. The two sets of experiments were performed on different HyperContext hyperspaces.

From the first set of experiments, we concluded that using interpretation- in conjunction with a weighted scale of confidence generally performed significantly better than the control experiments at recommending a relevant node which had not been previously encountered in the context session. Indeed, the control experiments generally recommended a node that had already been encountered in the context session.

In the second set of experiments, we used two different approaches to automatically search through an adaptive HyperContext hypertext and a non-adaptive hypertext. The adaptive approaches used interpretation- to derive a salient interpretation of the current document in context, and compared the salient interpretation to the description of the target document in order to recommend a path to the target document. The non-adaptive approaches re-used the interpretation of documents in the context bottom whenever a document was accessed. The interpretation of a document in the context bottom was

compared to the interpretation in the context bottom of the target document to recommend a path. We concluded that although using a link ordering algorithm (based on the similarity of a link's destination document to the target document) in the non-adaptive hypertext is the most successful overall, these successes occur mainly when the target document is a grandchild of the root document. When the target document is further away from the root document, the adaptive approaches are more successful.

The final question left to answer is whether the model of a user's short-term interests can sufficiently accurately represent the user's short-term interests, while the user is browsing through an adaptive HyperContext hypertext, to recommend a document which is relevant to the user. Ideally, this is answered by monitoring the experiences of users interacting with the HyperContext prototype. However, as explained in Section 9.5.1, this is not currently possible. Instead, we randomly generate a number of context sessions using the adaptive hypertext. At the end of each context session we derive two user models. The first user model, $UM_{adaptive}$, is derived using interpretation-. The second user model, $UM_{control}$, is derived using control. Both user models are derived in conjunction with the weighted scale of confidence. Each user model is then used to generate a user query and the first relevant document found, which has not already been encountered in the context session, is recommended to the user. The user is asked to give a relevance judgement for each recommended document.

The experiments are described further in Section 9.7, together with a presentation of the results.

9.7 Recommending documents using AID

Eleven paths through the HyperContext hypertext described in Chapter 8 are randomly selected. Each path is exactly 5 nodes long. The first node in each path is the hypertext's root document www.w3.org/Overview.html in the context bottom. A subsequent link is randomly selected from the interpreted document's out-links, and the destination document is accessed in context. If the destination of a randomly selected link has already been accessed during the same context session, we reject the link and randomly select another one. This process continues until either an interpretation with no out-links is reached, or once the path is five nodes long. If the final path is less than 5 nodes long, the entire path is rejected. We chose paths of length 5 to be consistent with the maximum lengths of paths which were tested during the experiments to determine which strategy was best at locating relevant information (see Section 9.5). Some documents used in the experiment occur in more than one path.

Once a path of length 5 has been selected, two user models $UM_{adaptive}$ and $UM_{control}$ are automatically generated, using interpretation- and control (Section 9.4.3) respectively to create the salient interpretations for each accessed interpretation. The salient interpretations are then weighted according to the scale of confidence (Chapter 5.8.2 and Section 9.4.3) and averaged to derive each user model (using formula 5.5). The eight terms with the highest weights are extracted from each user model to generate the user queries, and the queries are submitted to SWISH-E ([48], described in Chapter 8.4), where they are executed against a centralised inverted index of all interpretations of all documents in the hyperspace. The search results are examined to extract the highest ranking document which has not already been visited during the context session. If either query fails to identify an unseen relevant document, the path is discarded. If both queries identify the same unseen relevant document, the path is also discarded.

Each path of five documents together with the two recommended documents are packaged into a set, and are hosted on a Web server. All the links in each document are removed, and the next document in the path is accessed through a button which is displayed at the end of each document. A CGI script traces a user's progress through the path of five documents. When the user has reached the last document in the path, she is informed that the next document is the first recommended document. After the user has read this document, she is asked to give a relevance judgement, and, once she has given her feedback, she is then shown the second recommended document. She is once again asked to give a relevance judgement for the second recommended document, after which she is thanked for her participation and invited to evaluate another set of documents.

Although a user is told that the recommended documents have been recommended by different methods, he cannot tell whether the first or the second document has been recommended by $UM_{adaptive}$. In some sets, the $UM_{adaptive}$ recommendation is shown first, and in others it is shown second.

1	Highly relevant
2	Quite relevant
3	Quite non-relevant
4	Highly non-relevant

Table 9.15: Relevance judgements

One of four relevance judgements can be awarded to each recommended document (table 9.15). It is possible for both the document recommended by the adaptive user model and

the one recommended by the control user model to be awarded identical relevance judgements.

At the start of a new evaluation session, the session is given a unique session, or user, number, by incrementing the value of the last evaluation session. The set to be evaluated is also automatically assigned, by incrementing the value of the last set that was evaluated. This means that if the last session identifier was 51, and the set that was evaluated in that session was set 7 (corresponding to the seventh path), then the next session will be assigned the identifier 52 and set 8 will be evaluated, regardless of whether the same user had participated in session 51 or any other session. It is possible for the same user to evaluate the same set on more than one occasion. The user is not told which set she is evaluating.

Members (both students and staff) of the B.Sc. (Hons.) Information Technology degree at the University of Malta were invited by electronic mail to participate in the evaluation, which was conducted from October 12th, 2000 to November 6th, 2000. The evaluations were conducted on-line. We did not request personal information from participants, and we did not record the number of actual users who participated. The data captured by the CGI script consists of the session, or user, number, the set under evaluation, the document number in the path, the relevance judgement (if appropriate), and a time stamp (table 9.16).

22 11 1 Thu Oct 12 23:23:08 MET DST 2000
22 11 2 Thu Oct 12 23:26:07 MET DST 2000
22 11 3 Thu Oct 12 23:27:58 MET DST 2000
22 11 4 Thu Oct 12 23:29:58 MET DST 2000
22 11 5 Thu Oct 12 23:31:20 MET DST 2000
22 11 6 Thu Oct 12 23:32:34 MET DST 2000
22 11 7 3 Thu Oct 12 23:36:13 MET DST 2000
22 11 0 3 Thu Oct 12 23:37:26 MET DST 2000

Table 9.16: A sample set evaluation log file

The last entry in table 9.16 (which has 0 for the document number in the path) is a record of the relevance judgement given to the previous document (the second recommended document). Similarly, the penultimate record is a request for document 7 of set 11 (the second recommended document) together with the relevance judgement (3) given to the previous document which was the first recommended document.

The time stamp is recorded for a number of reasons. Initially, we wanted to be able to identify if a user had "rushed" through the set without actually reading the documents, so that the evaluation could be eliminated as invalid. Subsequently, as we processed the

results, we realised that significantly different time lapses were recorded against the same documents in different sessions, and in different sets. There are many possible interpretations for a short reading time. The two likeliest interpretations which can affect the interpretations of the results are that the document may previously have been read by the user while she was previously evaluating a set, or that the reader is skim reading through the set. We are reasonably confident that we are able to distinguish between the two interpretations. If a short time lapse is associated with just one or two documents in a set, and those documents occur in other sets, then we assume that the evaluator previously read the document while evaluating another set. On the other hand, if all the documents in a set have a short associated time lapse, then we assume that the evaluator is skim reading through the set[5]. We concluded that if a set is skim read, then the evaluator would have a surface understanding of the content, otherwise the evaluator would have a deep understanding of the content[6].

57 evaluation sessions were recorded. Of these, 30 were invalidated as the evaluator did not complete the evaluation. The results of the 27 valid evaluations are provided as an aggregate (table 9.17).

The overall results are inconclusive, mainly due to the small number of evaluations. Most of the sets have only 2 evaluations, and only two sets have 4 evaluations. Overall, the document recommended by $UM_{control}$ is considered more relevant than the one recommended by $UM_{adaptive}$ (48%). The document recommended by $UM_{adaptive}$ is favoured in only 22% of the evaluations. In the remaining 30% of the evaluations, the $UM_{adaptive}$ recommendations are considered as relevant as the $UM_{control}$ recommendations. This is quite significant, if we bear in mind that care was taken to ensure that control and adaptive recommended different documents. In tables 9.18 and 9.20 we break down the results to show the effect of displaying one of the recommended documents before the other, and the effect that skim reading appears to have on the results.

---

[5] There is the possibility that the user previously evaluated this set, or that all documents in the set were encountered by the user while evaluating other sets, but we still consider this set to have been skim read.
[6] Of course, if the evaluator is a speed reader, then these conclusions are invalid. Although we note the possibility that some of the evaluators may be speed readers, we discount it in our interpretation of the results.

Set No.	No. of Evaluations	$UM_{adaptive} > UM_{control}$	$UM_{control} > UM_{adaptive}$	$UM_{adaptive} = UM_{control}$
1	2	1	0	1
2	4	0	4	0
3	2	0	1	1
4	4	2	2	0
5	3	0	3	0
6	2	0	1	1
7	3	2	0	1
8	1	0	0	1
9	2	1	0	1
10	2	0	2	0
11	2	0	0	2
Total	27	6	13	8

Table 9.17: Aggregated results of the AID evaluation

Whether the document recommended by $UM_{adaptive}$ or by $UM_{control}$ was shown first appears to be significant (table 9.18). The $UM_{control}$ recommendation is overwhelmingly favoured if it is shown first (64%). This drops to 31% when $UM_{control}$ is shown second.

	$UM_{control}$ shown first	$UM_{adaptive}$ shown first
$UM_{adaptive} > UM_{control}$	3	3
$UM_{control} > UM_{adaptive}$	9	4
$UM_{control} = UM_{adaptive}$	2	6
Total	14	13

Table 9.18: Performance according to which recommended document was shown first

On six occasions when the $UM_{adaptive}$ recommendation was shown first, the $UM_{control}$ recommendations were considered equally relevant. However, on only one of these occasions was it impossible for the evaluator to give the $UM_{control}$ recommendation a better relevance judgement (because the evaluator had already given the $UM_{adaptive}$ recommended document a relevance judgement of "Highly Relevant").

	Skim read	Deep read
$UM_{adaptive} > UM_{control}$	2	4
$UM_{control} > UM_{adaptive}$	9	4
$UM_{control} = UM_{adaptive}$	3	5
Total	14	13

Table 9.19: Performance according to skim or deep reading a document set

The amount of time spent reading a set of documents was also significant (table 9.19). The quickest reading time for a set was 47 seconds, whilst the longest recorded reading time was 34 minutes and 6 seconds. We compared the actual reading time for each set of documents to the sum of the average reading time per document in each set. There was usually a large disparity between the actual and the average reading times. We consider a document set to have been skim read if the actual reading time was less than the average, otherwise we consider the set to have been deep read.

Roughly half the document sets evaluated were skim read. Of these, the $UM_{control}$ recommended document was given a better relevance judgement 64% of the time. This again drops to just 31% when the document sets are deep read. When the document set is read deeply, the number of times that the document recommended by $UM_{adaptive}$ is at least as relevant as the document recommended by $UM_{control}$ almost doubles.

	$UM_{adaptive}$ shown first		$UM_{control}$ shown first	
	Skim read	Deep read	Skim read	Deep read
$UM_{adaptive} > UM_{control}$	1	2	1	2
$UM_{control} > UM_{adaptive}$	3	1	6	3
$UM_{control} = UM_{adaptive}$	2	4	1	1
Total	6	7	8	6

Table 9.20: Performance according to skim or deep reading, and document order

The breakdown of skim or deep reading according to which recommended document was shown first also showed a consistent trend against favouring $UM_{control}$ when the document sets were deep read (table 9.20).

Although the overall results are inconclusive, one point in HyperContext's favour is that whereas the results for $UM_{control}$ cannot change, those for $UM_{adaptive}$ may change significantly if interpretations of documents are created by HyperContext's user community.

Finally, although the results do not necessarily reflect this observation, we are interested in the possibility of HyperContext and HyperContext-like adaptive hypertexts supporting not one, but two, models of a user's short-term interest, the first based on $UM_{adaptive}$, and the second based on $UM_{control}$. Apart from noticing that which recommended document was given a higher relevance judgement appeared to be influenced by whether the document set was skim read, we also noticed significant differences between the user models which were constructed in each case. In the case of $UM_{control}$, the query extracted from the user model reflected terms which tended to be very visible in the documents in the set (for example, terms which occur frequently throughout each document, or which occur in the documents' titles or headings). On the other hand, the query extracted from the user model based on $UM_{adaptive}$ tended to contain terms which required a closer inspection of the documents to see why they were selected. These differences in the user models, and the corresponding relevance judgements given to the recommended documents, suggest that depending on whether a document has been skim read or read thoroughly, a user's interest in the document may be different and would therefore require different descriptions.