Abstract

Current generation of hypertext systems suffer from the limitations that they are static in nature, and they do not support the automated process of link creation very well. Because of the efforts involved in manually creating links, the hyperbases created using these systems are seldom modified even when they were found not to fully support the requirements of the intended users. This paper studies the development of automated tools to aid the process of link creation, browsing, and link refinement. Only relation links are considered in this study. The automated tools are developed to help in three of the major stages of developing and using hypertext applications: (a) during authoring to generate a set of relation links between pairs of nodes; (b) during browsing to recommend an optimal set of starting nodes for the users to begin browsing, and to guide the users at each stage of browsing by suggesting a set of “next” nodes to traverse; and, (c) during training to modify, remove and add links based on users' feedback data collected. The training will result in long-term modifications to the hypertext structure.

In order to test the effectiveness of the training process objectively, a navigator is built to simulate the browsing activities of the users. The effects of training have been evaluated on two text collections using a variety of objective measures. The results indicate that the training process has improved the effectiveness of the hyperbase to support browsing.

Keywords: Hypertext; Browsing; Links; Information Retrieval; Relevance Feedback
1. Introduction

Hypertext offers a new approach to organising and accessing large amount of unstructured information. It encodes information as a collection of independent nodes that are connected through links. It provides the users a great deal of flexibility and freedom in traversing the links and accessing the information network. However, the flexibility, coupled with the inherent complexity of the hypertext network structure, also give rise to the problem of "dis-orientation" when accessing large hyperbases\(^9\). The user gets dis-oriented when he does not know where to start, where he is, where he is heading to, and/or how he may reach a known location in the hyperbase. One approach to alleviate this problem is to guide the user during the browsing process by providing recommendations on the optimal starting points to begin browsing, and the "next" nodes to traverse during each stage of browsing.

However, providing better browsing support is only part of the solution, the static nature of the current hypertext systems also contributed to the disorientation problem in many ways. This is because for such systems, the entire network structure is fixed. Even if there is a mis-match in the concept structure between the users and the hyperbase, it is not possible for the system to alter its structure in order to suit the users. Such mis-match will occur even in well-designed hyperbases because it is not possible to anticipate all users' requirements especially if the users' requirements are also changing over times. The mis-match will render the users unable to locate useful information. Thus, one would like the hypertext to be dynamic, so that the structure of the hyperbase may be altered automatically during usage to suit the users.

In addition, a useful hypertext system must also be able to handle effectively the vast amount of information available in traditional form. The conversion of linear text to hypertext involves the tedious tasks of: (a) segmenting the text into self-contained nodes; and (b) creating links between the related nodes\(^14,16\). Of the two, the link creation has been found to be the more time-consuming and laborious task. If links are to be created and maintained by different authors over a long period of time, inconsistencies in the way in which the nodes are linked would arise. Such inconsistencies would aggravate the dis-orientation problem of accessing the hyperbase.

To address the "disorientation" problem, we need a powerful computation engine that could aid in the process of link creation, browsing, and link refinement. The computation engine needs to unify two approaches of information access: non-procedural query and procedural browsing. The problems of automatically generating hypertext links have been considered by many researchers\(^1,3,7\). Almost all approaches use information retrieval (IR) techniques\(^26\) as the basis to assign links between semantically related nodes. Most systems consider the problems of generating relation links, which is defined as the bi-directional link that relates the contents of two nodes as a whole. Examples of relation links are the "see also" references in the UNIX manual. In terms of browsing support, one system that attempts to return an optimal starting point for the user to start browsing in a hierarchical hypertext network is reported in
Frisse. The approach can be extended to a general hypertext network to provide guidance to the users during browsing.

There is, however, no reported work on using the users' feedback information to modify the links and hyperbase structure automatically. Such automated support will be extremely useful in providing a dynamic system that could respond to users' changing information needs. Much progress has been made in the field of IR to utilise users' relevance feedback data to enhance the retrieval performance of the IR systems. In particular, effective relevance feedback techniques have been developed to modify user's query and/or node descriptions in order to effect both short term and long terms improvements in retrieval effectiveness. In addition, the use of spreading activation technique to modify link weights during browsing has also been reported. Such techniques can be applied to the hypertext domain to modify link structures.

The aim of this paper is to investigate the development of a computation engine to provide automated support for link creation, browsing and link refinement. In particular, the use of relevance feedback techniques from IR and the spreading activation technique in AI as the basis to modify hyperbase structure will be investigated. In line with other studies, only relation links are considered here. The automated tools are developed to help in three of the major stages of developing and using hypertext applications: (a) during authoring to generate a set of relation links between pairs of nodes; (b) during browsing to recommend an optimal set of starting points for the users to begin browsing based on their free-text queries, and to guide the users at each stage of browsing by suggesting a set of next nodes to traverse; and, (c) during training to modify, remove and add links based on users' feedback data collected. In order to test the effectiveness of the training process objectively, a navigator is developed to simulate the (sometimes random) process of browsing by the users. The effects of training have been evaluated on two text collections of over 80 and 120 nodes respectively. Objective measures such as the recall value of the retrieval and the compactness of the relevant node set are used as the basis for comparison. The general results indicate that the training process has improved significantly the effectiveness of hypertext system to support browsing.

The contents of this paper are as follows. Section 2 provides the background of related research in the area of IR and hypertext. Section 3 discusses the design of automated tools and algorithms to support link creation, browsing, and link refinements. Section 4 describes the implementation of the hypertext system incorporating these tools. In Section 5, we discuss the design of a navigator to simulate the users' browsing activities. Section 6 discusses the evaluation procedures and the results of applying users' relevance feedback data to train the hyperbase. Finally, Section 7 presents the conclusions.

2. Background
In this section, we review the free-text retrieval techniques for the IR systems and navigation techniques employed in hypertext systems. A good discussion of the application of IR techniques to hypertext systems can be found in Savoy\textsuperscript{27}. In addition, we will discuss suitable evaluation measures for hypertext systems.

### 2.1 Free-Text Information Retrieval Techniques

Information Retrieval (IR) is concerned with the representation, analysis and retrieval of text nodes based on user's queries\textsuperscript{26}. Currently, most IR systems use single term as basic indexing unit. The use of single term IR techniques has been found to be surprising effective in retrieving a large proportion of relevant nodes, and in rejecting a large proportion of non-relevant nodes\textsuperscript{23}. Given a free-text node, a simple text analysis algorithm is employed to: (a) extract individual words from the node; (b) remove common words stored in a stop-list; and (c) stem the remaining terms using, say, the Porter's stemming algorithm\textsuperscript{21}. The result is a vector of word stems. In order to represent the relative importance of each stem term in the node, a term weighting scheme is used to assign numerical weight to each term. A high performance term weighting scheme assigns large weights to terms that occur frequently in particular nodes, but rarely in the rest of the nodes. A normalised version of such term weighting scheme is defined as\textsuperscript{20,24}:

$$w_{ij} = \frac{tf_{ij} \cdot idf_{j}}{\sqrt{\sum_{k=1}^{t} (tf_{ik} \cdot idf_{k})^2}}$$  \hspace{1cm} (1)$$

where

$$idf_{j} = \log \left( \frac{N}{n_{j}} \right)$$

where $w_{ij}$ and $tf_{ij}$ represent respectively the weight assigned to and frequency of term $j$ in node $i$; $N$ is the total number of nodes in the collection; $n_{j}$ denotes the number of nodes in which term $j$ appears; and the denominator is the Euclidean length of the term vector of node $i$. $idf_{j}$ is generally known as the inverse document frequency for term $j$.

The resulting term weight has a value of between 0 and 1 and is independent of the size of the node. Given a node vector $D_{i} = (w_{i1}, w_{i2}, .., w_{it})$ and a query vector $Q_{j} = (q_{j1}, q_{j2}, .., q_{jt})$, the similarity between $D_{i}$ and $Q_{j}$ is given by:

$$Sim(D_{i}, Q_{j}) = \sum_{k=1}^{t} W_{ik} \cdot q_{jk}$$  \hspace{1cm} (2)$$
The Sim values computed are used as the basis to return a ranked list of nodes to the users. The similarity measure given in Equation (2) can also be used to compute the similarity between two nodes, and has been used as the basis to compute relation links.

Because of the ambiguity of many of the terms used in the node descriptions and queries, the IR techniques, though effective in retrieving a large proportion of relevant nodes, are unable to achieve a high retrieval precision of over 60%. Two main approaches are employed in an attempt to improve the retrieval performance. One is through the judicious use of domain knowledge, usually in the form of domain-specific thesaurus\textsuperscript{10,28}, and the other is to use users' relevance judgments of the earlier retrieval to perform relevance feedback operation\textsuperscript{25}. The use of domain knowledge is problem dependent and require expensive hand-crafting of domain knowledge for the technique to be effective. On the other hand, relevance feedback operations can be performed automatically as soon as the users' relevance judgment data is available. A typical formula used to modify user's query using relevance feedback technique is given by\textsuperscript{5}:

$$Q^{(i+1)} = Q^{(i)} + \alpha \sum_{D \in R} D - \beta \sum_{D \in NR} D$$

(3)

where $Q^{(i+1)}$ is the modified query of $Q^{(i)}$, $R$ and $NR$ are respectively the sets of relevant and irrelevant nodes judged by the users through earlier retrieval iterations; and $\alpha$, $\beta$ are the bias factors.

The modified query is then re-submitted to the system for retrieval. While query re-formulation is effective in improving retrieval performance for a single user's request, it does not initiate long-term improvements to the system. Document space modification techniques address this issue by applying Equation (3) to modify the node's contents directly to make relevant nodes more easily retrievable in the future and irrelevant nodes less so. Experimental results indicate that this technique is effective in providing permanent improvements to the performance of the text-based system. However, the effectiveness of the technique relies heavily on a close match between the text queries used to update the node descriptions, and the subsequent queries issued by the users\textsuperscript{5,22}. In cases where there is a mis-match between the two types of queries, the performance of the system might degrade. This problem is partially caused by the use of “conflicting” data to “train” (or update) the system.
2.2 Link Creation and Browsing Support in Hypertext System

2.2.1 Automated Link Creation

There are many types of links used in the hypertext systems. The three most common generic link types used are the reference link, relation link and structure link. Reference links are used to relate part or whole of a node with one or more other nodes. It is usually uni-directional and are generally used to create cross-referencing structure commonly found in almost all hypertext systems. Relation links are used to relate nodes that address similar subject matters. They express a strong semantic relationship between two nodes as a whole and are usually bi-directional. Structure links are used to express the hierarchical relationship between a parent node and its children, thus forming a tree within the hypertext structure.

Link creation is a major task in the conversion of linear text to hypertext. A common strategy is to extract links from structured documents or mark-up text. In Justus, reference links are automatically created within structured legal documents by utilising the tight coupling between semantic markers and well-defined reference phrases in legal documents. Furuta et al. extract reference links from documents that have previously been marked up using SGML. HyperNews uses the structured header of the network news articles to create reference links to other related articles. These techniques work well for structured text, but they are not easily extendable to un-structured text as useful reference information is usually obscured within the text.

Other projects consider the problems of creating relation links in un-structured text. Bernstein developed a simple apprentice that uses the IR similarity matching technique to propose potential new content links to the authors. IR techniques are also used in HEFTI and HyperNews to generate relation links between semantically related text. Others use concept-based IR technique to generate both reference and relation links in unstructured text. Experiments carried out in these works showed that a large proportion of links discovered using IR techniques are reasonably relevant.

2.2.2 Browsing Support

Browsing is a heuristic search through flexible organisation of multimedia nodes in order to find information relevant to one's need. Browsing has been perceived as a manual task and has been a primary concern of user interface designer. However, intelligent supports for browsing are required in at least two areas: to help users get started in a large hyperbase, and to guide the users along a preferred path.

The integration of query-based IR technique into hypertext system to support browsing was first investigated by Frisse in the Medical Handbook Project. Frisse modifies the traditional IR technique to take into consideration the presence of links to suggest optimal starting points for browsing in a hierarchical hypertext network. Given a user's query, Frisse assigns weight to a node based on two
components: (a) the intrinsic weight which depends only on its contents; and (b) the extrinsic weight which depends on the weights of its descendants. The total weight of node i is therefore:

\[ W_i = IW_i + EW_i = \sum_{j=1}^{i} w_{ij} + \frac{1}{y} \sum_{k \in S_i} IW_k \]  

(4)

where \( IW_i \) and \( EW_i \) are respectively the intrinsic and extrinsic weights of node i; \( S_i \) is the set of descendants of node i and \( y \) is the size of set \( S_i \). The term weights \( w_{ij} \) are assigned using a variant of term weighting scheme defined in Equation (1).

The propagation function (4) is called recursively from leaf nodes to the root node. The optimal starting point for browsing is the node with the highest weight. A similar technique can also be used to suggest a list of "next" nodes for the users to traverse during browsing. However, the choice of optimal next nodes needs to also take into account the current browsing context.

Other approach considers the use of spreading activation technique\(^\text{19}\) to modify the link weights and suggest candidate nodes to explore next. In spreading activation, nodes have different levels of activation. A node with a higher activation level would have a higher possibility of being a useful node to view next. When a node i is activated with its activation value increased by \( AV_i \), it passes this increment by spreading activation to all its neighbours. Each neighbour receives an activation level in proportion to its link weight with node i, as expressed in the following formula:

\[ AV_j = \mu \cdot AV_i \cdot \left( \frac{LW_{ij}}{\sum_{k \in S_i} LW_{ik}} \right) \]  

(5)

where \( LW_{ij} \) is the link weight between nodes i and j; \( S_i \) denotes the set of neighbours of i; \( y \) is size of set \( S_i \); and \( \mu < 1 \) is the damping factor for the spreading activation.

The process of spreading activation is recursive and it continues until the activation value drops below a certain threshold or when all paths have been traversed. The damping value \( \mu \) is used to ensure that as the activation spreads further away from the source, the activation value will decrease geometrically and thus ensuring that the process will stop. Since Equation (5) depends only on the fan-out factor of a node, the time required to perform spreading activation is independent of the size of the network. This makes it an efficient technique for updating the links' weights in large hyperbases.

Connectionist formalism\(^\text{2}\) suggests another approach to support hypertext browsing and information retrieval. This approach views the inverted index of the nodes as a set of opposing directed link pairs between the nodes and the index terms. During browsing, user's response to a specific node would propagate activities to other nodes and index terms. After activation, browsing candidates would be ranked and presented to the user.
Croft and Turtle\textsuperscript{11}, on the other hand, view the inverted index as basic representations of node relations. Hypertext links provide additional evidence about the nodes' relations and hence the inverted index are modified to take the links into consideration. The modified inverted index is then used to support querying within the hyperbase. Because modifications of inverted index are needed when link structure are altered, this approach seems more adopted to static hypertext environment\textsuperscript{27}.

2.3 Evaluation of Hypertext System

Evaluation on the usability of the hypertext system to support user browsing and information retrieval is an extremely difficult task. Extensive testings on a wide range of user community and task types should be carried out before any useful conclusion can be drawn\textsuperscript{18,30}. This is clearly beyond the scope of this investigation. Fortunately, various objective structural measures to deduce the complexity of the hyperbase have been studied\textsuperscript{4}. Our assumption here is that a network that is structural too complex will not be usable, the inverse is not true of course. The use of recall measure in IR\textsuperscript{26} to test the effectiveness of a retrieval can also be employed. All measures to be reviewed assume that the set of relevant nodes for the current retrieval session is known before hand.

One measure to evaluate the ease in locating all relevant information in the hypertext network is to compute the average cost of locating the first relevant node, \( F \), and subsequently retrieving the rest of relevant nodes, \( S \):

\[
Cost = F + S = F_i + \sum_{j \in R_i} \text{path}_{ij}
\]

where \( F_i \) is the cost of locating the first relevant node \( i \) in the relevant node set; \( \text{path}_{ij} \) is the path length from node \( i \) to node \( j \); and \( R_i \) denotes the rest of relevant set excluding node \( i \). If a computation engine is used to suggest an optimal node to start browsing, then \( F = 0 \) if the node returned is a relevant node; otherwise, \( F \) is the path length from the top node to the first browsed relevant node. Once the first relevant node \( i \) is located, \( S \) is simply the total path length from node \( i \) to the rest of the relevant nodes. If not all the relevant nodes can be reached through browsing (because some nodes are not connected to the relevant set), then \( S \) is set equal to the total number of nodes in the system. Hence Equation (6) is a measure of the number of link traversals needed to retrieve all relevant nodes in the hyperbase.

One useful measure to assess the quality of hypertext network is to compute the compactness value\textsuperscript{4} of the relevant set. Although compactness of the overall hypertext network is not very meaningful\textsuperscript{4}, the compactness of the relevant set is useful. If the relevant set is compact, it implies that the relevant nodes are well connected to each other, and hence can be easily retrieved via browsing once the first relevant node is found. Otherwise, it implies that some of the relevant nodes might not be connected to the relevant set, suggesting that not all relevant nodes may be reached via browsing alone. Following Botafogo et al\textsuperscript{4}, the compactness of the relevant node set within the hyperbase is defined as:
\[
C_p = \frac{(\text{Max} - \sum_i \sum_j C_{ij})}{(\text{Max} - \text{Min})}
\]

(7)

and \( \text{Max} = (n^2 - n)C \); and \( \text{Min} = (n^2 - n) \).

where \( C_{ij} \) is the path length between nodes i and j in the relevant node set; C is the maximum possible path length between two nodes in the hyperbase; and n is the total number of relevant nodes. Here we set C equal to the total number of nodes in the hyperbase. This compactness value varies between 0 and 1, independent of the number of nodes. It gives a value of 0 when the node set is completely disconnected and 1 when it is fully connected.

Recall is one of the popular measures used to evaluate the effectiveness of the IR system\textsuperscript{26}. Recall measures the fraction of relevant nodes in the collection that are retrieved. The same measure can be extended to the context of hypertext browsing to measure the fraction of relevant nodes that were located during a browsing session. The formula is given below:

\[
\text{Recall} = \frac{\text{Total number of relevant nodes retrieved}}{\text{Total number of relevant nodes in the hyperbase}}
\]

(8)

It should be noted that in order to compute the evaluation measures expressed in Equations (6-8), we need to know in advance the set of nodes that are relevant to the search goal in each retrieval session. For simplicity, the search goal is expressed using a free-text query, and the relevant set is predefined by the authors before the retrieval, and is known as the "predefined relevant set". This is the common approach employed in the evaluation of IR systems, where standard test collections are used that include not only the text nodes and the query set, but also the pre-defined relevant set for each query.

3. The Design of Computation Engine

In this section, we describe the design of automated tools to be used in our computational engine to support the process of link creation, browsing and link refinement. The detailed algorithms employed in these tools are also described here. As mentioned in the previous section, we will consider only relation links in this paper.

3.1 Automatic Generation of Relation Links

As demonstrated in HyperNew\textsuperscript{1} and Bernstein's Apprentice\textsuperscript{3}, a large proportion of relation links discovered using the standard IR techniques is reasonably relevant. Hence, our link creation module will use a similar approach to generate relation links. Here, we use the IR scheme defined in Equations (1-2) to compute the similarity between each pair of nodes. For N nodes in the hyperbase, an NxN SIM matrix is first computed, where SIM(i,j) stores the similarity value between nodes i and j. The SIM matrix is
symmetrical and hence only the upper (or lower) half of the matrix needs to be stored. The SIM matrix is used as the basis to generate relation links and to determine the validity of links during training (see section 3.3).

Given that we want to generate an average of $k$ links per nodes, the total number of links to be generated is therefore $K = N \times k$. From the SIM matrix stored, a threshold value can be found by first sorting all non-zero entries in the SIM matrix into a list, and use the value of the $K^{th}$ entry as the threshold. This threshold value is used as the cut-off point to determine whether there is a valid link between two nodes. A valid link exists between two nodes $i$ and $j$ if $\text{SIM}(i,j) = \text{threshold}$. However, this simple approach of determining valid links between nodes may give rise to isolated nodes or nodes having too many links. From the usability point of view, nodes having too many links are undesirable as it is confusing to the users as to which link to traverse. On the other hand, isolated nodes are not reachable from other nodes via browsing. Hence, we need to establish further constraints on the minimum (MIN) and maximum (MAX) number of links permitted for each node during initial link generation.

The following algorithm generates an average of $k$ links per node using the stored SIM matrix subject to the MIN and MAX constraints:

```
For row i=1...N in the SIM matrix {
    Let p be the number of valid links assigned to row i so far;
    Sort the rest of non-zero entries in row i in decreasing order of link weight
    and store them in sortTable;
    Let q be the number of entries in sortTable;

    // to impose Minimum Link Constraint
    if (p+q) < MIN {
        Randomly generate additional (MIN-p-q) links along row i;
        Mark these links as valid links;
    }
    else
        Mark the top (MIN-p) entries in sortTable as valid links;
        if (weight of a valid link ij) < threshold
            Set SIM(i,j) = threshold;

    // generate additional valid links up to the maximum constraint
    While ( ( not end of sortTable ) AND
            ( link weight of current entry > threshold ) AND
            ( number of links < MAX ) ) {
        mark the current entry as valid link;
    }
}```
advance to next entry in sortTable;
    }
}

// ** Note that the maximum link constraint may be violated for certain nodes

After the execution of this algorithm, only those links that are marked valid are used for browsing. These valid links have their corresponding SIM matrix values set equal to or greater than the threshold value. The validity of the links may be updated by the training process.

3.2 Support for Browsing

As discussed previously, intelligent supports for browsing are required in at least two areas: (a) to help users find a good starting point to begin browsing; and (b) to guide users at each stage of browsing.
3.2.1 Selecting Optimal Starting Point

To locate an optimal starting point for browsing, we use an approach similar to that used in the medical handbook project\(^\text{13}\). However, Frisse's approach was designed for a hierarchical hypertext system, hence some modifications are needed to adopt Frisse's approach to a general hypertext network. One major refinement made in our approach is that instead of considering all nodes as possible starting point, we limit it to the set of nodes whose contents are sufficiently similar to the search goal (or user's query). This is because for a non-hierarchical network, it is not meaningful and confusing to suggest a starting node whose content has little resemblance to the user's query.

Based on the user's initial query, our system first computes a ranked list of possible starting nodes using the IR scheme described in Equations (1-2). The top \( r \) nodes are selected and stored in the initial ranked list, \( R^0 \). In order to increase the probability of selecting a useful starting node from \( R^0 \), we modify the weights of the nodes in \( R^0 \) by adding in the contributions from their neighbours. The final weight of a node \( i \) in \( R^0 \) is defined as:

\[
W_{t_i} = \text{Sim}(D_i, Q) + \frac{1}{y} \sum_{j \in S_i} \text{Sim}(D_j, Q)
\]

where \( S_i \) is the set of neighbours of node \( i \); and \( y \) is the size of set \( S_i \). \( R^0 \) is re-ranked based on the modified weights and the top node is returned to the user for browsing. If the top node is found subsequently not to be useful, the next best node is returned upon user's request.

3.2.2 Selecting the Next Node

To prevent the user from "getting lost" in the maze of information, the system provides recommendations of what appears to be a promising direction to head given the user's current state. A ranked list of "next nodes" is computed from the neighbours of the current node and returned to the user for selection. The next node list is computed by taking into consideration the query and the user's current browsing state. Nodes that have been browsed will not be in the next node list, and nodes that are known to be in the original ranked list \( R^0 \) are given higher probabilities of being selected. Thus for efficiency reason, a simple formula is used to compute the weight of a next node \( j \) with respect to the current node \( i \):

\[
W_{t_j} = \sigma \cdot \text{Sim}(i, j)
\]

where \( \sigma \) is the bias factor which is set to a value greater than 1 when the next node is found in the original ranked list \( R^0 \); otherwise it is set to 1. The list of next nodes is then sorted in descending order of \( W_{t_j} \) and presented to the user for evaluation.

3.3 Feedback To Update Hyperbase Structure
As discussed previously, it is not possible for the hyperbase designer to anticipate all users' requirements, especially if the users' requirements are changing over time. This is even more so for automatically generated link structure as it is likely that a high percentage (over 30%) of links generated are irrelevant. This is because the basic IR techniques used in link generation are not of sufficiently high accuracy in determining node similarities. This problem is tackled in most systems by requiring the authors to manually modify the links generated. An alternative approach is to collect feedback information from the users while using the system, and use this information to modify the link structure automatically.

In section 2.2, we have reviewed the use of spreading activation and document space modification techniques to update the contents of the hyperbase. The idea here is to use similar techniques to update link weights, and to add and delete links as a result of changes in the link weights. Throughout the discussion here, we assume that the search goal of the retrieval can be expressed using a free-text query, and that the search goal does not change in the course of browsing, unless it is explicitly requested by the users to modify the query (see Section 3.4). The assumption is needed to provide a basis for evaluating and updating links.

To illustrate the training or link updating process, consider part of a hyperbase during a browsing session as shown in Figure 1. The nodes in Figure 1 are classified into three types: the current node, previous node and neighbour node. Current node is defined as the node that the user is currently viewing; while previous nodes are those that the users have previously viewed and have provided relevance judgements. It should be noted that there might not be a direct valid link between the current node and the previous nodes. Neighbour nodes on the other hand are the immediate neighbours of current node that have not been viewed by the users. In this example, node 4 is the current node; nodes 1, 2 and 3 are the previous nodes; while nodes 5 and 6 are the neighbour nodes. Figure 1 also shows the relevance judgements on nodes 1, 2 and 3 with respect to the search goal.

![Figure 1: Part of hypertext network during a browsing session](image-url)
First we need to determine whether positive or negative activation should be performed between the current node and each of the previous nodes. The decision, of course, depends on whether the current node is being judged as relevant or irrelevant to the search goal. If the current node is judged to be relevant, then from Figure 1, positive activations should be performed between current node and nodes 1 and 3, and negative activation is performed between the current node and node 2. On the other hand, if the current node is judged to be irrelevant, then only negative activation is performed between current node and nodes 1 and 3. Activation between two irrelevant nodes is not performed.

Once the type of activation is determined, a 2-level positive (or negative) spreading activation is then performed. As an example, suppose that the current node 4 has been judged as relevant and positive activation is to be performed between nodes 4 and 3. The activation is performed in 2 steps. First the link weight between nodes 4 and 3 is incremented by a positive activation constant $AV_c$. Next this activation level is propagated proportionally among the neighbour nodes to make them more closely linked with previous node 3. In this example, the link weights between nodes 3 and 5, and between nodes 3 and 6 will be incremented. The amount of increment is defined using Equation (5). In a similar way, if negative activation is to be performed between current node and node 2, then the corresponding link weights will be decremented accordingly.

It should be noted that the link updating is performed on the SIM matrix directly. If as a result of a positive update, the link weight of a previously invalid link becomes greater than the threshold value, then the link will become valid. On the other hand, the link weight of a previously valid link may become less than the threshold as a result of a negative update, then such link will be marked as invalid. In this way, we are able to add or remove links as the training proceeds.

The rational behind the training is to make relevant nodes more similar or strongly linked with each other, and less similar with the irrelevant nodes. This applies also to their neighbours because if node A is strongly linked to node B, then the neighbours of node A should also be more closely linked with node B based on the definition of relation link, unless there is evidence to suggest otherwise. Because of the updates on the neighbour nodes, nodes that were not directly connected are also affected, and hence new links may be "added" as a result.

### 3.4 Query Modification

In hypertext, browsing is both an information seeking as well as learning process. Through the process of browsing, the user learns more about the contents (and structure) of the hyperbase and is thus able to formulate a better query. Thus after some browsing, if the user feels that he is unlikely to get sufficient relevant nodes, he will attempt to re-formulate the query based on the nodes that he has browsed. Hopefully, a better starting point can be found so that more relevant nodes can be retrieved.
Our system supports this process by performing automatic query modification based on user's feedback data. After the user submits an initial query, the system assists the user in browsing through the network of nodes. From the nodes traversed, the user evaluates their relevance and feedback to the system. Upon user's request, the system reformulates the query using the formula defined in Equation (3). The modified query is then resubmitted to the system to compute a new set of starting points for browsing.

4. The Implementation of Hypertext Training System

This section briefly describes the design and implementation of the hypertext training system (HTS). This system is designed to enable the users to browse through the hyperbase, provide relevance judgement on the nodes traversed, and activate the system to update the link structure. The organisation of HTS is shown in Figure 2. It consists of 5 major modules, corresponding to the 5 automated functions that the system is designed to provide. The modules are: (a) Link Generator; (b) Start Node Adviser; (c) Next Node Adviser; (d) Link Trainer; and (e) Query Processor. These modules act as intermediaries between the users and the hyperbase.

![Figure 2: The Structure of the Hypertext Training System (HTS)](image)

Given a set of text nodes, the Link Generator is first called to build the NxN SIM matrix; and to generate an average of k links per node using the link generation algorithm described in section 3.1. The validity of the links and their final link values are stored in the SIM matrix, which is residing in the hyperbase.

The Start Node Adviser is used to compute a ranked list of possible start nodes. When a start node is requested by the user, this adviser returns the node at the top of the list that has not been chosen before. If this list is exhausted, the adviser rebuilds another set of start nodes. At each stage of browsing, the Next Node Adviser is activated to build a ranked list of next nodes for the users to select.
At any time during training, the user may provide relevance judgments on the nodes traversed with respect to the query. This information, together with the sequence of nodes browsed, are stored in the browsing history list. The list is used by the Link Trainer to modify the link weights in the SIM matrix.

Finally, the Query Processor is used to provide query-based search function, and to modify the query automatically based on users' feedback data. The new query can be used to initiate another browsing session.

The system is implemented on Silicon Graphics Indigo system. Only a simple user interface is built to enable the users to perform link traversal and relevance feedback operations. The system is implemented using C++ and the interface is built using InterViews.
5. The Design of a Navigator to Simulate Users' Browsing Activities

In this section, we discuss the techniques for evaluating the system. A thorough evaluation of the system requires careful monitoring of the use of the system by real users on a wide range of tasks. The process is very tedious and requires careful control of the experimental process. In order to facilitate the evaluation process, we develop a navigator to simulate the behaviour of the users when using the system. The Navigator employs a random function to simulate the process of selecting a suitable node to browse during each stage of browsing. The Navigator assumes that the relevant node set of the current retrieval session is known, and hence the relevance of a node being browsed can be determined automatically.

In order for the simulation to be realistic, users' decisions on two major browsing tasks must be properly modelled. Users' decisions are constantly needed to determine: (a) the link to traverse during each stage of browsing; and (b) whether to pursue the current branch or to back-track if the current node is not relevant. In addition, the maximum attention span of the user during a browsing session needs to be considered. For simplicity, we simulate the attention span of the users by the maximum number of nodes to be browsed in one session. For our simulation, we fixed this number to be 15 as we feel that it is a reasonable number of nodes that a user will evaluate for a given query.

To simulate users' decision making process on link selection, we employ a random function to select a link to pursue from the "next node" list suggested by the system. Since the next node list is ranked, for new users, it is likely that they will select a next node that is ranked high in the list. This suggests that the probability of a node being selected is directly proportional to its weight in the next node list. For experience users, however, they will also give preference to nodes whose names are similar to the search goal. This can be simulated by giving higher probabilities to those nodes that are very similar to the query and appear in the initial ranked list $R^0$ (see 3.2.2). To combine these two factors, we perform the followings on the nodes in the next node list: (a) for those nodes that also appear in the $R^0$ list, we increase their weights by multiplying them by a positive constant greater than 1 (in our case 1.2); (b) next, we normalise the resulting weights of all nodes by dividing them by the total node weight; and (c) we assign an unique range of between 0 and 1 to each node in the next node list. A random number generator is then used to generate a random number of between 0 and 1. If the random number falls within the range of a particular node, then that node will be selected as the next node.

Next, we need to simulate the users' decision on back-tracing. A user will consider back-tracking if: (a) the current and previous nodes browsed are not relevant; and (b) the current node is too dissimilar to the query. The second reason for back-tracking can be simulated by checking whether the similarity between the current node and the query is below a threshold, MinSimilarity. Also, an impatient user might decide to back-track to a previously found relevant node as soon as an irrelevant node is encountered. This last rarer behaviour can again be simulated using a random number generator.
The algorithm to simulate user's browsing behaviour can now be described. The algorithm uses the following variables:

- **NoLinksTraversed**: counts the number of links traversed (or node browsed) so far;
- **MaxSpan**: defines the maximum number of nodes to browse per session;
- **HistoryList**: keeps track of the list of nodes browsed and the relevance judgments;
- **Sim(currNode, Query)**: returns the similarity between the current node and the query using Equation (2);
- **MinSimilarity**: defines the minimum acceptable similarity between the current node and the query before a back-tracking is to be performed;
- **GetStartNode()**: returns the next highest ranking start node computed by the Start Node Adviser;
- **GetNextNode(currNode)**: returns the next node to browse from the current node using the simulation procedure discussed previously.

```python
NoLinksTraversed = 0;
HistoryList = NULL;
CurrNode = GetStartNode();
while (NoLinksTraversed < MaxSpan)
{
   // Evaluate CurrNode
   store CurrNode and its Relevance status in HistoryList;
   increment NoLinksTraversed by 1;

   if (CurrNode is a relevant node)
      // proceed by choosing a link to traverse from CurrNode
      CurrNode = GetNextNode(CurrNode);
   else
      if (Sim(CurrNode, Query) < MinSimilarity) {
         // similarity too low, unlikely to be fruitful to pursue this branch
         CurrNode = BackTrack(CurrNode);
      }
      else {
         if (previous node is not relevant)
            BackTrack(CurrNode);
         else {
            // Randomly decide whether to pursue this branch
            // Give higher probability to pursue
            if (to pursue)
               CurrNode = GetNextNode(CurrNode);
         }
      }
}
```
else BackTrack(CurrNode);
}
}
}

and the algorithm for the BackTrack(CurrNode) function is:

BackTrack(CurrNode) {
    if (HistoryList is not empty AND a relevant node can be found)
        CurrNode = last relevant node found;
    else
        CurrNode = GetStartNode();
}

At the end of the browsing simulation, the Link Trainer may be called to update the link structure using the procedure outlined in Section 3.3.

6. Results and Evaluation

6.1 The Training Procedure

In order to evaluate the effects of training on a wide variety of situations, including the modifications of queries and hyperbases, three distinct phases of training are performed on each query. The Navigator is employed in each phase to simulate the users’ browsing behaviour. At the end of each phase, performance measures are computed and recorded. The performance measures collected are: (a) the cost of retrieving all relevant nodes; (b) the compactness of the relevant node set; and (c) the recall of relevant nodes. They are defined in Section 2.3. For convenience, we will refer to them simply as the Cost, Compactness and Recall respectively throughout this section. The first two measures provide indications on the quality of links within the relevant node set, while Recall gives the effectiveness of the retrieval process in locating as many relevant nodes as possible.

Let the hypertext that we start with be $H^{(0)}$, and the original query issued be $Q^{(0)}$, the training phases can be described as:

Phase A:
Browse hyperbase $H^{(0)}$ using query $Q^{(0)}$ and collect the browsing history in $B^{(0)}$. Use $B^{(0)}$ and $H^{(0)}$ to compute the performance measures. The purpose of this phase is to collect performance measures before
the training for used as the basis of comparison. Note that even without training the Cost of retrieving all relevant nodes may be different as different browsing session may take a different path.

**Phase B:**
Train the hyperbase $H^{(0)}$ based on the browsing history $B^{(0)}$ to obtain a new hyperbase $H^{(1)}$. Browse hyperbase $H^{(1)}$, again, using query $Q^{(0)}$ and collect the resulting browsing history in $B^{(1)}$. Compute the performance measures based on $B^{(1)}$ and $H^{(1)}$. The purpose of this phase is to evaluate the effects of training without query modification.

**Phase C:**
Modify the query $Q^{(0)}$ using the relevance data derived in $B^{(1)}$ to obtain a modified query $Q^{(1)}$. At the same time, train the hyperbase $H^{(1)}$ based on the browsing history $B^{(1)}$ to obtain a new hyperbase $H^{(2)}$. Now browse the hyperbase $H^{(2)}$ using query $Q^{(1)}$, and collect the browsing history in $B^{(2)}$. Compute the performance measures for this phase based on $B^{(2)}$ and $H^{(2)}$. The purpose of this phase is to evaluate the combined effects of both query modification and training on modified hyperbase.

In order to simulate the effects of updating the system by different users, we repeat the three-phase training process 10 times, each time starting from the original hyperbase $H^{(0)}$ and query $Q^{(0)}$. The browsing history for each round is different because of the random processes of selecting the next node to browse and determining whether back-tracing should be performed. The performance statistics collected at the end of 10 rounds are averaged for evaluation purposes.
6.2 Profile of Database

Two text collections are used to evaluate the effectiveness of relevance feedback and training on the hypertext system. Both text collections include a set of queries with corresponding pre-defined relevant nodes for each query. Table 1 summarises the characteristics of the two text collections.

<table>
<thead>
<tr>
<th>Text Collection</th>
<th>SG</th>
<th>ADI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of documents</td>
<td>121</td>
<td>82</td>
</tr>
<tr>
<td>Number of words</td>
<td>15,967</td>
<td>5,294</td>
</tr>
<tr>
<td>Number of words per document</td>
<td>132</td>
<td>65</td>
</tr>
<tr>
<td>Number of stemmed words</td>
<td>2,626</td>
<td>1,030</td>
</tr>
<tr>
<td>Number of queries</td>
<td>30</td>
<td>38</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of text collections

The first collection, SG, is a set of 121 text nodes in the domain of Singapore history. It is developed using our earlier hypertext system\(^8\). It includes 30 queries. Each query having between 1 to 10 relevant nodes. The average number of relevant nodes per query is 3.3. These queries were selected to have at least one term indexed and at least one term not indexed in the original text nodes. This means that some relevant nodes are likely to be retrieved on the initial query to the unmodified hyperbase but probably not all of them. Thus these queries could potentially benefit from the training process.

The second text collection, ADI, is a public-domain data set widely used in the IR community to test the effectiveness of the IR systems\(^12\). It is included here to provide a more objective test of our system. It consists of 82 text nodes and 38 queries with pre-defined relevant node set for each query.

6.3 Experiment Results

For each test collection, we build two hyperbases with an average of 2 links (SG-2 & ADI-2) and 4 links (SG-4 & ADI-4) per node. The reason for building two hyperbases with different number of average links per node is to evaluate the effects of training on systems with different number of inter-connections. For each hyperbase, we carry out the three phases of browsing and training as described in Section 6.1 ten rounds. The results of each phase are then averaged and tabulated for evaluation in the next 2 subsections.
6.3.1 Results and Evaluation of the SG collection

Table 2 tabulates the average Cost, Compactness and Recall over all the queries for SG-2 and SG-4 at each phase.

<table>
<thead>
<tr>
<th>Phase</th>
<th>SG-2</th>
<th>SG-4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>Compact</td>
</tr>
<tr>
<td>A</td>
<td>48.8</td>
<td>0.57</td>
</tr>
<tr>
<td>B</td>
<td>38.9</td>
<td>0.63</td>
</tr>
<tr>
<td>C</td>
<td>24.7</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Table 2: Performance statistics of SG Text Collection

From Table 2, we observe that the results of SG-2 and SG-4 follow similar trends as we progress from phase A to phase C. In both cases, the Cost of retrieving the relevant nodes are decreasing, while the Compactness and Recall of the relevant set are increasing, hence demonstrating that the application of relevance feedback and training has improved substantially the effectiveness of the hyperbase to support browsing.

To show the percentage improvements of performance at the query level, we collect in Table 3 statistics on the percentage of queries whose performance measures have improved over the phases.

<table>
<thead>
<tr>
<th>Condition</th>
<th>SG-2</th>
<th>SG-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of queries whose cost is improved from A to B</td>
<td>53%</td>
<td>34%</td>
</tr>
<tr>
<td>% of queries whose cost is improved from A to C</td>
<td>73%</td>
<td>47%</td>
</tr>
<tr>
<td>% of queries whose compactness is improved from A to B</td>
<td>24%</td>
<td>12%</td>
</tr>
<tr>
<td>% of queries whose compactness is improved from A to C</td>
<td>36%</td>
<td>16%</td>
</tr>
<tr>
<td>% of queries whose Recall is improved from A to B</td>
<td>38%</td>
<td>48%</td>
</tr>
<tr>
<td>% of queries whose Recall is improved from A to C</td>
<td>57%</td>
<td>58%</td>
</tr>
</tbody>
</table>

Table 3: Percentage improvements in Performance Measures for SG Collection

From Table 3, we observe that a high percentage of queries have their Cost reduced and Compactness increased after the training on the hyperbases. However, a much higher percentage of queries in SG-2 registers such improvements than in SG-4. As the Cost and Compactness give indications on the quality of the links within the relevant node set, these results show that the quality of links has improved more
substantially in SG-2 than in SG-4. This is to be expected as SG-4 is better connected at the beginning of
the experiment and hence there is less room for improvement.

Although high percentage of queries also register increase in their Recall values in both hyperbases, the
percentage of increase is more substantially in SG-4 than in SG-2. This is rather surprising but can be
explained as follows. Because SG-2 contains less number of links, its relevant node set is less well
connected. Hence there is less opportunity for the browsing operation to reach all the relevant nodes to
improve their associations. In the case where irrelevant nodes are retrieved, the only action that the
system can do is to reduce the opportunity of retrieving these nodes in future, and hopefully, the relevant
nodes will surface. On the contrary, the predefined relevant set in ADI-4 is better connected and hence
easier for the browsing activity to reach the relevant nodes. Once these relevant nodes are retrieved, the
connectiveness of the predefined relevant set can then be improved, making it even easier to retrieve
other relevant nodes in future. This finding seems to suggest that with intelligent supports, it is better to
have more links in each node to ensure that the information retrieval process via browsing is more
effective.

The results also show that there is no degradation in performance measures for all queries.

6.3.2 Results and Evaluation of ADI collection

For the ADI collection, we tabulate the corresponding results for ADI-2 and ADI-4 as shown in Tables 4
and 5.

<table>
<thead>
<tr>
<th>Phase</th>
<th>ADI-2</th>
<th></th>
<th></th>
<th>ADI-4</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>Compact</td>
<td>Recall</td>
<td>Cost</td>
<td>Compact</td>
<td>Recall</td>
</tr>
<tr>
<td>A</td>
<td>50.2</td>
<td>0.26</td>
<td>62.8</td>
<td>42.3</td>
<td>0.98</td>
<td>69.6</td>
</tr>
<tr>
<td>B</td>
<td>41.9</td>
<td>0.54</td>
<td>65.3</td>
<td>34.6</td>
<td>0.98</td>
<td>81.3</td>
</tr>
<tr>
<td>C</td>
<td>25.6</td>
<td>0.72</td>
<td>80.3</td>
<td>18.4</td>
<td>0.99</td>
<td>88.3</td>
</tr>
</tbody>
</table>

Table 4: Performance statistics of ADI Text Collection
<table>
<thead>
<tr>
<th>Condition</th>
<th>ADI-2</th>
<th>ADI-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of queries whose cost is improved from A to B</td>
<td>55%</td>
<td>25%</td>
</tr>
<tr>
<td>% of queries whose cost is improved from A to C</td>
<td>63%</td>
<td>52%</td>
</tr>
<tr>
<td>% of queries whose compactness is improved from A to B</td>
<td>36%</td>
<td>32%</td>
</tr>
<tr>
<td>% of queries whose compactness is improved from A to C</td>
<td>42%</td>
<td>38%</td>
</tr>
<tr>
<td>% of queries whose Recall is improved from A to B</td>
<td>17%</td>
<td>43%</td>
</tr>
<tr>
<td>% of queries whose Recall is improved from A to C</td>
<td>33%</td>
<td>48%</td>
</tr>
</tbody>
</table>

Table 5: Percentage improvements in Performance Measures for ADI Collection

In can be seen from Tables 4 and 5 that the results for the ADI collection are very similar to those observed in SG collection. Table 5 again shows that a much higher percentage of queries improved their Recall for ADI-4 as compared with ADI-2 after training, thus confirming our earlier finding that training is more effective for better connected hyperbases.

6.4 Main Findings of the Experiment

To conclude, we observe that the results from SG and ADI collections are rather similar. The hyperbases generated using both collections improve their retrieval performance substantially when relevance feedback technique is used to train the hyperbase and reformulate the queries. These results are encouraging and demonstrate that the training process is effective.

As to the hyperbases with different average number of links per node, we found that when the number of links is lower, there is more opportunity to use training to improve the quality of links of the relevant set. However, the inverse is true for Recall, which increases more substantially when there are more links. As Recall measures the percentage of relevant nodes found via browsing, one can conclude that training is more effective when there are more links in the system. This finding, however, is preliminary as we are still yet to test the effectiveness of the system when there are more than 4 average links per node, and on large hyperbases of over 1,000 nodes.

7. Conclusions

In this paper, we describe the design and implementation of a hypertext training system (HTS). The system is designed to provide automated supports for the process of link creation, browsing, and link
refinement. Given a set of independent text nodes, the system is able to generate a set of relation links automatically using the standard IR techniques. The average number of links per node generated is controllable by the users. During browsing, the system will provide recommendations on a good starting node to begin browsing, and a ranked list of next nodes to view at each stage of browsing. The recommendations are computed using a modified IR technique that takes into consideration the presence of links. The browsing process also collect relevance judgments provided by the users. The relevance judgments are used during the training stage to modify the link weights of the hyperbase. A 2-level spreading activation technique is used to modify the link weights between the nodes browsed, and their neighbours. As a result of these updates, new links may be added while some links may be removed. Hence, the structure of the hyperbase is able to change in response to users' feedback on the system.

The system is tested using a Navigator built to simulate the users' behaviour when using the system. A three-phase training process is designed to train the system under a variety of situations. Many training runs are performed to simulate the use of the systems by different users. The results on two text collections indicate that the training process has improved significantly the effectiveness of the hyperbase to support browsing. The results also show that training is more effective in improving the retrieval effectiveness when there are more links in the system. The findings seem to suggest that with intelligent supports, it is better to have more links in each node to ensure that the information retrieval process via browsing is more effective.

One main obstacle to the application of the automated techniques to large hyperbases is the large amount of storage and processing required to maintain the SIM matrix. However, an examination of the training process reveals that the SIM matrix is required only during link creation and the subsequent updating of links. The computation of optimal node set to support browsing using Equations (9-10) involved only the set of valid links. If the valid links are stored together with the set of nodes, then little additional storage is needed as compared with normal hypertext system to support browsing. Thus, one strategy to handle large hyperbases is to treat link updating (or training) as an off-line batch process. Since the speed of link updating is not critical, the SIM matrix can be stored on disk and only loaded one row at a time when needed. Efficient indexing structure needs to be devised to support this process of accessing SIM matrix. More research is needed to investigate the feasibility of this strategy.

Further research is also needed in the following areas: (a) to study the effects of adding more links and long-term training on the usability of the system; (b) to consider automated supports for other link types, such as the reference or structure links; (c) to conduct tests on large scale text collections of over 1,000 nodes; and (d) to conduct usability tests on real users.

Acknowledgments
The authors would like to thank the National University of Singapore for the provision of a research grant #RP920637 under which this research is carried out.

Author's Note

All correspondence regarding this paper should be directed to Tat-Seng Chua at the address given.

References


