Pergamon

0306-4379(94)E0004-9

Information Systems Vol. 19, No. 4, pp. 33–54, 1994
Copyright © 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/94 $7.00 + 0.00

ADAPTIVE CLUSTERING OF HYPERMEDIA DOCUMENTS

ANDREW JOHNSON[1], FARSHAD FOTOUHI[2]

[1]Electronic Visualization Laboratory, University of Illinois at Chicago, Chicago, IL 60607, USA
[2]Wayne State University, Dept. of Computer Science, Detroit, MI 48202, USA

(Received 18 June 1993; in final revised form 1 November 1993)

Abstract — A hypermedia system connects various types of information into a network where related nodes of information (text, audio, video) are connected by links. Clustering these nodes is an effective way to reduce information-overhead, allowing the user to browse through the clusters as well as the individual nodes. In this paper, we compare the use of two adaptive algorithms (genetic algorithms, and neural networks) in clustering hypermedia documents. These clusters allow the user to index into this overwhelming number of nodes and find needed information quickly. We base the clustering on the user's paths through the hypermedia document and not on the content of the nodes or the structure of the links in the document, thus the clustering reflects the unique relationships each user sees among the nodes. The original hypermedia document remains untouched, however each user will now have a personalized index into this document.

Key words: Hypermedia, Indexing, Clustering, Genetic Algorithms, Neural Networks

1. INTRODUCTION

A hypermedia system connects information into a graph structure where related nodes of information are connected by links [3] [16] [18]. A node (or information element) is a piece of information. A node can contain text, a picture, a sound, a piece of animation, or a combination of the above. Hypermedia systems designed for browsing through inter-related information are supposed to make it easier for their users to access related data. Unfortunately the very flexibility of these systems causes the user to become lost in a maze of information elements.

This confusion can be reduced by clustering the nodes into higher level concepts, and then allowing the user to index into the hypermedia document using these clusters. Current clustering in hypermedia systems rely too heavily on the author. While the author sets up the nodes and links in the hypermedia document, it is the user who needs the benefits of clustering, so the user should be involved in the clustering process. Each user sees different relationships between the nodes. The hypermedia system should allow the users to mold the hypermedia document to their particular needs. This flexibility will be increasingly important as special purpose hypermedia documents give way to more general purpose hypermedia documents. The incredible increase in the number of WWW home pages over the last couple years with multimedia information makes this kind of clustering even more important as information can be easily, and repeatedly accessed by a very large number of people. Our solution is to cluster based on the user's navigation through the hypermedia document. The content of the nodes is irrelevant.

We collect data on the user's journeys through the links. This information is then given to a genetic algorithm and to a neural network which partition the nodes of the hypermedia document into clusters. The genetic algorithm and neural network generate very good clusterings much faster than a deterministic algorithm. The user now has the choice of browsing through the network of nodes in the original hypermedia document, or using a hierarchy of clusters as an index to move quickly to the appropriate node in the network. The original hypermedia document remains untouched, however each user now has a personal index into this document. See Figure 1.

As new users begin to work with this hypermedia document, they can choose from the existing clusterings. A new user can choose to look at the hypermedia document from various points of view. This gives each user a starting point near their own needs. Experienced users will also be able to access their data faster and more conveniently because the hypermedia document will adapt to them.

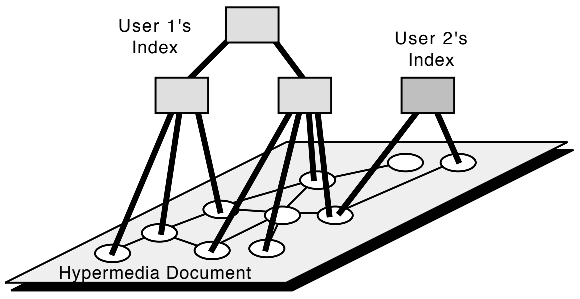

Fig. 1: Hierarchical Index into the Hypermedia Document

We have used a similar technique to collect information on a user's querying of a scientific database to cluster the tables based on the user's usage patterns [14, 13]. The hypermedia user, like a scientific database user, is faced with an overwhelming amount of interrelated information presented in a generic way. Clustering allows us to add a personal interface layer on top of the generic interface reducing information overhead and speeding up access to the underlying data.

Section 2 discusses how we collect information from the user. Section 3 discusses clustering in hypermedia documents. Section 4 discusses how we use a genetic algorithm to create the hierarchy of clusters. Section 5 discusses how we use a neural network to create the hierarchy of clusters. Section 6 discusses our implementation of both clustering methods and compares the two methods. Section 7 discusses how these clusters can be used. Finally, Section 8 gives our conclusions and plans for future work.

2. INFORMATION COLLECTION

We want to see what the user is doing without bothering the user. The user wants to browse through the hypermedia document to get answers. The user is less likely to use the system if extraneous duties are added on. The simplest way to do this is to keep track of the paths the user takes through the hypermedia document - the system simply looks over the user's shoulder.

Several ideas of tracking user behavior are based on keywords or descriptors. These keywords or descriptors are grouped into concepts. The keywords for each node can be defined by the author, or the hypermedia system can look for occurrences of words in the nodes to define concepts. The user chooses which concepts he is interested in and the hypermedia system starts the user off in the appropriate region of the hypermedia document [2], [5], [7].

With hypermedia documents being created in many countries these keywords can be in several languages. New systems deal with sound and video so these keywords do not exist. Either accurate meta-information must be explicitly stored (giving the author more work to do) or we must try something else. Given that a majority of work on the world wide web is being created by 'amateur' authors, it would be unreasonable to expect all these people to add additional meta-information into their multimedia documents. In our system the actual information stored in each node is unimportant. Instead of concentrating on the content of the nodes, we concentrate on the paths the user takes through the nodes. The previous systems require their users to make decisions affecting the promotion and demotion of keywords during their search. We feel this is an unacceptable burden on the average user. We believe a truly useful information gathering process must work without the user's direct intervention.

A node contains one or more links to other nodes. These links connect the nodes into a network. The simplest form of link moves the user from one node to another. A link is activated by selecting an anchor (a word, picture, or special icon in a node.)

We can model a hypermedia document as a directed graph as shown in Figure 2 where:

Roman letters (A, B) represent node IDs. We assume that each node in the hypermedia document is given a unique ID so the node ID labels a vertex of the graph. Each node contains 0 or more source anchors.

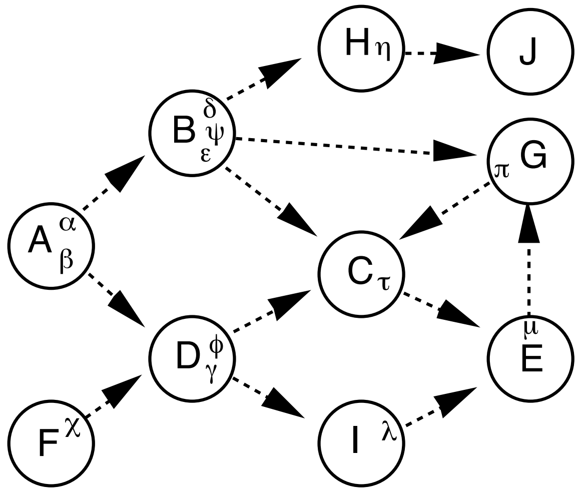

Fig. 2: Graphical Representation of a Hypermedia Document

$\int\rightarrow A\beta\rightarrow D\gamma\rightarrow I\lambda\rightarrow E\rightarrow\int$
$\int\rightarrow A\beta\rightarrow D\phi\rightarrow C\tau\rightarrow E\rightarrow\int$
$\int\rightarrow A\alpha\rightarrow B\delta\rightarrow H\eta\rightarrow J\rightarrow\int$
$\int\rightarrow F\chi\rightarrow D\gamma\rightarrow I\rightarrow\int$

Fig. 3: The Paths a User Takes through the Hypermedia Document

Greek letters (α, β) represent source anchor IDs. We assume that each anchor in a node is given a unique ID so the combination of the node ID and source anchor ID label an outgoing edge of the graph.

The user creates a path (a sequence of nodes visited and anchors activated) while browsing through a particular hypermedia document. A sample set of 4 paths are shown in Figure 3 where:

$\int$ represents the hypermedia system (where each path begins and ends.).

So, for example, in the first path the user moves from the hypermedia system to node A, then activates anchor β in node A to move to node D, then activates anchor γ in node D to move to node I, and so on.

We can break these paths up into their individual 'node anchor $\rightarrow$ node' transitions and store them graphically as shown in Figure 4 where:

Arabic numerals (1, 2) represent the number of times that a source anchor has been activated.

So, for example, in the 4 paths of Figure 3 the user twice used anchor β to move from node A to node D, so the link from node A to node D using anchor β is given weight 2. The user once used anchor α to move from node A to node B, so the link from node A to node B using anchor α is given weight 1. The user never used anchor π to move from node G to node C, so the link from node G to node C using anchor π is given no weight.

We can store the set of paths that a user has taken through a hypermedia document in a list structure, as shown in Figure 5.

Each element of the list has the form: $CNID : \{PNID(\{CAID.weight - NNID\#\}^{+}),\}^{+}$

CNID - Current Node ID
PNID - Previous Node ID
CAID - Current Anchor ID
NNID - Next Node ID
weight - # of times CAID was used.

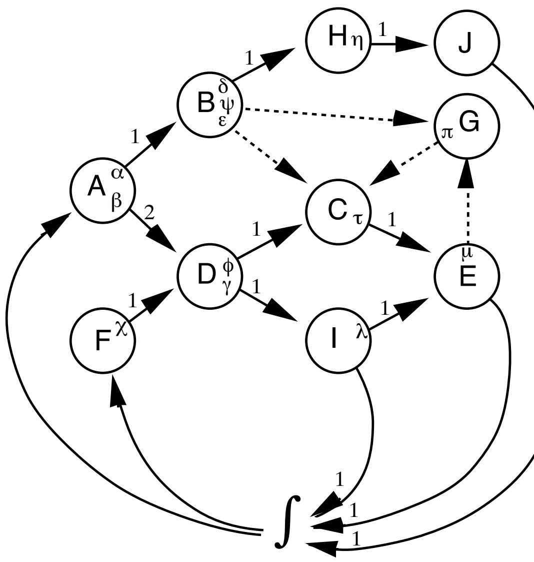

Fig. 4: Graphical Representation of the Paths

A:	$\int(\alpha.1\text{-B}\#\beta.2\text{-D}\#)$,
B:	A(δ.1-H#),
C:	D(τ.1-E#),
D:	A(γ.1-I#ϕ.1-C#), F(γ.1-I#),
E:	C($\int$.1-$\int$#), I($\int$.1-$\int$#),
F:	$\int$(χ.1-D#),
H:	B(η.1-J#),
I:	D(λ.1-E#$\int$.1-$\int$#),
J:	H($\int$.1-$\int$#),

Fig. 5: List Representation of the Paths

So, for example the line "A: $\int(\alpha.1\text{-B}\#\beta.2\text{-D}\#)$," shows that if the user is at node A (current node), and came to node A from the hypermedia system $\int$ (previous node), then one time the user activated anchor α (current anchor) in node A to move to node B (next node), and two times the user activated anchor β (current anchor) in node A to move to node D (next node).

Using the previous node as well as the current node to give a weight to each anchor has several advantages over storing full paths, or simply using the current node to give a weight to an anchor. A user browsing through a hypermedia document may be thinking about one concept or several concepts. Since we do not want to bother the user by asking each time the user changes concepts it would be inappropriate to store full paths in the system as each full path may contain several concepts. Our system of using the current node and previous node to give a weight to each anchor also allows the mixing of paths that have one link in common. This allows different paths to merge. However simply using the current node to give a weight to each anchor goes too far in isolating the nodes from the paths that led the user to visit them. Using only the current anchor, any paths that have a common node will be linked together and the path information will be lost. Our method of using the previous node and the current node keeps the important path information while allowing the paths to merge.

When a user begins to search for information in the hypermedia document they either choose to work with an existing list or create a new empty list. The user can give this new empty list an appropriate name. As the user works with the hypermedia document, the list keeps track of the nodes the user is accessing through their browsing.

A user can create one list for each hypermedia document, or create multiple lists for the same

Node	# Times Used	Neighbours
A	3	BD
B	1	AH
C	1	DE
D	3	ACFI
E	2	CI
F	1	D
H	1	BJ
I	2	DE
J	1	H

Fig. 6: Conversion of a Single List

hypermedia document where each concept the user is interested in is given its own list. The lists for a single hypermedia document can be unioned together. A list can therefore be one of a user's lists, the union of a user's lists, the union of a group's lists, or the union of a site's lists. A more thorough discussion of the information collection process, and other applications are given in [12].

3. CLUSTERING

As hypermedia documents grow larger it becomes more and more important to cluster their nodes. In 1987, Halasz [10] discussed 'composites' and 'virtual structures' as two important issues for the next generation of hypermedia systems. Clustering reduces information overhead. It allows for higher level concepts (groups of nodes), allows the breaking of a single large hypermedia document into appropriate modules, allows for views over the hypermedia document, and allows the user to make changes at the cluster level without affecting the hypermedia document itself. For more information on clustering see [4] [8].

Current clustering in hypermedia systems is either structurally driven where the connectivity of the nodes delineates the clusters [1] [11] or concept driven where the keywords (either in the nodes themselves or in the meta-information about each node) delineate the clusters [2] [6]. Structurally driven clustering does not take into account how the user navigates through the nodes and links. Concept driven clustering has difficulty if the hypermedia document contains foreign languages or non-textual items. Where are the keywords?

Both clustering methods require the hypermedia document to provide meta-information about itself. This information may not be available, or accessible. Both rely too heavily on the author.

By monitoring the user's progress through the document we can cluster the hypermedia document based on the connections the user makes between the nodes. Those nodes that the user has found important are made more visible, and more easily accessible. In commonly used documents this will give the user quick access to important blocks of information, and several starting points for starting new searches. The clusters can be shared between users, giving new users a choice of several ways to view an unfamiliar document, and giving regular users a way to see the document from different perspectives.

The lists that have been created by the information gathering process described in Section 2 are used to cluster a hypermedia document. The number of times each CNID is used in a list (i.e. the sum of all its anchor weights) is calculated. For each CNID, every associated PNID and NNID in each of the lists is added onto the roster of neighbouring nodes for that CNID. Figure 6 shows how the single list of Figure 5 is converted.

Converted information from all the lists to be used the clustering is then combined to generate the input to the clustering algorithms. The number of times each node is used in a single list is divided by the total number of times that node was used in all the lists to generate usage percentages. For each node, all of the neighbouring nodes are unioned together to form a single neighbourhood roster for that node.

The usage percentages show how important each node is to the various topics. A node may be important to one topic, a set of topics, or to all of the topics; clusters of nodes may be important

	List 1		List 2
A:	$\int(\alpha.1\text{-B}\#\beta.2\text{-D}\#)$,	A:	$\int(\alpha.1\text{-B}\#)$,
B:	A(δ.1-H#),	B:	A(ϵ.1-C#),
C:	D(τ.1-E#),	C:	B(τ.1-E#),
D:	A(γ.1-I#ϕ.1-C#), F(γ.1-I#),	E:	C(μ.1-G#),
E:	C($\int$.1-$\int$#), I($\int$.1-$\int$#),	G:	E($\int$.1-$\int$#),
F:	$\int(\chi.1\text{-D}\#)$,		
H:	B(η.1-J#),		
I:	D(λ.1-E#$\int$.1-$\int$#),		
J:	H($\int$.1-$\int$#),		

Fig. 7: Two Lists Used In Clustering

Node	List 1 %	List 2 %	Neighbours
A	75	25	BD
B	50	50	ACH
C	50	50	BDE
D	100	0	ACFI
E	66	33	CGI
F	100	0	D
G	0	100	E
H	100	0	BJ
I	100	0	DE
J	100	0	H

Fig. 8: Conversion of the Two Lists

to the same set of topics. The neighbourhood roster shows which nodes the user connected during their browsing.

As the number of lists increases, the usage percentages decrease towards zero for the commonly used tables. This decreases the effectiveness of the clustering algorithms as the amount of difference in the usage patterns decreases. To avoid this pitfall, the percentages are scaled so that the average usage percentage is set to 50% rather than (100/number of lists)%.

Figure 7 shows the list from Figure 5 and a second list to be clustered. Figure 8 shows these two lists converted into their usage percentages and neighbourhood rosters. For example, Node A is used 3 times in list 1 and 1 time in list 2 yielding 75% and 25% as the usage prcentages. Node A has neighbours B and D in list 1 and neighbour D in list 2 yielding B and D as the neighbour roster.

We use a genetic algorithm, and a neural network to delineate the clusters. Using a deterministic algorithm would require evaluating the quality of every partition of the set of nodes. This is an exponential problem (there are over 5 trillion different clusterings of 19 nodes) so it is too costly to find the highest quality clustering for a document with more than a few nodes. Using an adaptive algorithm such as genetic algorithms or neural networks we can find a very good quality clustering within a reasonable amount of time.

4. GENETIC ALGORITHM CLUSTERING

Genetic algorithms perform searching in a manner similar to natural selection in nature. By mimicking "survival of the fittest," genetic algorithms try to 'evolve' a solution to a search problem. A population of possible solutions is used. The more fit members of this population (i.e. those that are nearer to the solution) are more likely to mate and produce the next generation. As the generations pass, the members of the population should get fitter and fitter (i.e. closer and closer to the solution.)

Genetic algorithms accept their input coded as a finite length string (or chromosome.) Each

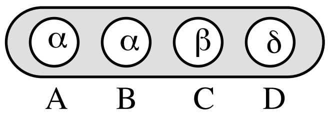

Fig. 9: Sample Chromosome

of the elements in the chromosome is a gene, and each gene has an allele value. The allele values are elements of a finite alphabet. The search problem is coded into the number of genes in a chromosome and the possible allele values. The space of possible solutions to the search problem is bounded by the number of permutations of genes that are possible.

A set of these chromosomes form a population. The population changes from one generation to the next through reproduction, crossover, and mutation. The number of chromosomes in each iteration remains constant, but their fitness should generally improve. The fitness of a chromosome is determined by a payoff function which depends on the allele values of each chromosome.

The 'goal' of the genetic algorithm is to produce a chromosome with the highest possible fitness; that is, the sequence of allele values that most closely matches the optimal point in the search space. The payoff function converts one chromosome into a single number. When the payoff function has been applied to all of the chromosomes their relative fitness can be determined.

Sometimes when two chromosomes reproduce, they crossover. When chromosomes W and X crossover to form chromosomes Y and Z, a certain set of chromosome W's genes are copied into identical positions in Y. The remaining genes in chromosome W are copied into Z. Genes from chromosome X fill in the remaining positions in Y and Z. That is, each of the children inherits some of the traits of each of their parents. Crossover only mixes existing allele values at each gene. Mutation causes random changes in the allele values, allowing for more variation.

For a more thorough introduction to genetic algorithms see [9].

In our approach, we use a population of 10 chromosomes. Each chromosome represents a set of clusters; each gene represents a node in the hypermedia document; each allele value represents a cluster. Genes with the same allele value are in the same cluster. Each node is a member of at most one cluster, and the number of clusters varies from 1 to n where n is the total number of nodes being clustered. Each node can form its own unique cluster in the chromosome, if necessary.

Figure 9 shows a sample chromosome that would be used to cluster four nodes. The four genes (A, B, C, D) in the chromosome represent each of the four nodes involved in the clustering. Nodes A and B are in the same cluster as they both have an allele value of α. Node C forms its own cluster with an allele value of β, and Node D forms its own cluster with an allele value of δ. As there are four nodes (A, B, C, D), there are four possible clusters (α, β, γ, δ.)

In addition to crossover and mutation we implemented two other operations to alter the chromosomes: fusion and fission. Fusion takes two unique allele values and combines them into a single allele value, combining two clusters into one. Fission takes a single allele value and gives it a different random allele value, breaking a cluster apart.

The percentage possibility of each operation is given below:

Fusion	75%
Crossover	50%
Fission	25%
Mutation	2%

These values were found experimentally to yield good overall results for a variety of problem sizes.

The payoff function is composed of two separate partial payoff functions: weight similarity, and neighbourlyness. Each partial payoff function's values range from 0.0 to 10.0.

Weight similarity, computed by Equation 1, promotes clusters containing nodes with similar weight percentages, indicating similar usage patterns. For every cluster, the difference between each node percentage and the average percentage of the cluster is computed and scaled into the range 0 to 10. The closer the percentages are, the smaller this ratio will be, and the larger the overall payoff will be. Weight similarity pushes the genetic algorithm to create smaller clusters.

$$10 \times (1 - min(\frac{\sum_{nodes} \sum_{lists} \mid node\% - aveCluster\% \mid}{\# \ of \ nodes}, 1)) \tag{1}$$

Neighbourlyness, computed by equation 2, promotes clusters containing neighbouring nodes. Each edge in the input lists must either connect two nodes in the same cluster, or two nodes in different clusters. Since the edge is in one of the lists at least one user must feel these two nodes are related because that user connected them. Neighbourlyness sums up the number of edges between nodes in the same cluster and divides the total by the total number of nodes in the lists. Neighbourlyness pushes the genetic algorithm to create bigger clusters.

$$10 \times \frac{\sum_{clusters} \# \ of \ edges \ between \ nodes \ in \ cluster}{\# \ of \ edgess} \tag{2}$$

Neighbourlyness tries to bring all the connected nodes together. Weight similarity tries to isolate nodes with similar usage patterns. Together, this pulling together and pushing apart generates the clusters. Equation 3 shows how the two partial payoff functions are combined. This payoff function was found to yield good results across a variety of problem sizes.

$$payoff = neighbourlyness \times weightSimilarity^2 \tag{3}$$

This results in an overall payoff function value that ranges from 0 to 1000.

Each node is initially put into its own cluster. The genetic algorithm is started. After each new generation is created, the most fit chromosome of the previous generation replaces the least fit chromosome of the new generation. This preserves the best set of clusters from one generation to the next. The genetic algorithm stops when the value of the most fit chromosome remains constant over 200 generations, or the number of generations reaches a set maximum value.

If there is more than one resulting cluster, these clusters are given back to the genetic algorithm. The weight percentages, and neighbours of the components of each cluster are used to generate the weight percentages, and neighbours for the cluster as a whole. Each cluster is now treated like a node. The clusters are clustered forming a hierarchy until one global cluster remains.

When the clustering is complete, the genetic algorithm will have placed all of the hypermedia-nodes into the leaf nodes of the cluster hierarchy. Now a breadth first traversal is made of the cluster hierarchy from its leaf nodes to the root. Each hypermedia-node is taken in turn. If it is a neighbour of all of the hypermedia-nodes in its own node and in each of its siblings' nodes then it is moved up one level in the tree. If this leaves an empty node with no children, then that node is removed. This way common hypermedia-nodes are moved up the tree through the internal nodes, and the depth of the tree is reduced.

5. NEURAL NETWORK CLUSTERING

A neural network is a parallel distributed network of small processing elements. This large number of simple processing units is connected together forming a miniature version of the human brain. Neural networks have had a good deal of success in pattern recognition and learning by example.

We are using a self organizing feature map, a neural network developed by Teuvo Kohonen [15]. A self organizing feature map is a two dimensional array of processing elements. Each element contains a weight vector. The input to the self organizing feature map is a set of weight vectors. The self organizing feature map forms itself into a topological ordering of the input data through unsupervised learning.

The weight vector W_{ij} of each element of the N by N array is initialized with random values. An input vector I is chosen at random. The neuron whose weight vector W_{ij} is closest to I is found $(min \mid W_{ij} - I \mid)$. This neuron and all the neurons within a certain neighbourhood (N_c by N_c) of this neuron have their weights adjusted, bringing the weights closer to I. Random inputs are repeatedly provided while the amount of adjustment (α), and the neighbourhood shrink in a form of simulated annealing. The amount of adjustment and the size of the neighbourhood eventually become negligible and the learning ceases.

When the learning stops, the network has organized the inputs into a two dimensional array. For each input there is one processing element with the closest matching weight vector. Labeling each neuron with the node that it matches most closely gives the topological ordering of the inputs. For a more thorough introduction to neural networks see [17].

In our clustering algorithm the weight vectors contain the usage percentages for the various usage lists being used to cluster the hypermedia document. If there are k lists involved in the clustering, the weight vector will contain k values. The size of the array N = $\lceil \sqrt{number\ of\ nodes}\ \rceil$. Each element can form its own unique cluster in the array, if necessary.

For each iteration in the learning process, one node is chosen at random. The neuron with the most similar set of weights is identified. That neuron and all the neurons within the current neighbourhood have their weights adjusted bringing them closer to the weights of the chosen node.

The weight vectors are adjusted as follows:

$$W_{ij_{t+1}} = \begin{cases} W_{ij_t} + \frac{\alpha_t}{2} \times (I - W_{ij_t}) & \text{if node is within } N_{c_t} \\ W_{ij_t} & \text{otherwise} \end{cases}$$

One neighbour of the chosen node is selected at random. The chosen node and all the neurons within the current neighbourhood have their weights adjusted bringing them closer to the weights of the selected neighbouring node.

The weight vectors are adjusted as follows:

$$W_{ij_{t+1}} = \begin{cases} W_{ij_t} + \frac{\alpha_t^2}{20} \times (I - W_{ij_t}) & \text{if node is within } N_{c_t} \\ W_{ij_t} & \text{otherwise} \end{cases}$$

The neighbourhood and the amount of adjustment (α) are then adjusted. If α is greater than zero, another iteration is performed.

α is adjusted as follows:

$$\alpha_0 = 1$$

$$\alpha_{t+1} = \alpha_t - 0.0001$$

the neighbourhood is adjusted as follows:

$$N_{c_t} = M \times \frac{\alpha_t}{2}$$

These values were found experimentally to yield overall good results for a variety of problem sizes.

After the array has been generated each node is taken in turn. Its set of weight vectors are compared to the elements of the array and the best match is found. Each array element is then labeled with the set of nodes claiming it as their best match. Thus all of the nodes are mapped to their appropriate spot in the array.

6. IMPLEMENTATION

We implemented this dynamic hierarchical indexing approach by creating a Hypercard stack called Control which monitors the user's browsing in another Hypercard stack and creates a list as discussed in Section 2. Control can then pass several of these lists to an external C function which runs one of the clustering routines discussed in Sections 4 and 5. The hierarchy of clusters is then given back to Control which allows the user to visualize, and index into the original stack through the hierarchy of clusters. The monitoring program is small enough that we can run concurrently with the hypermedia document being clustered.

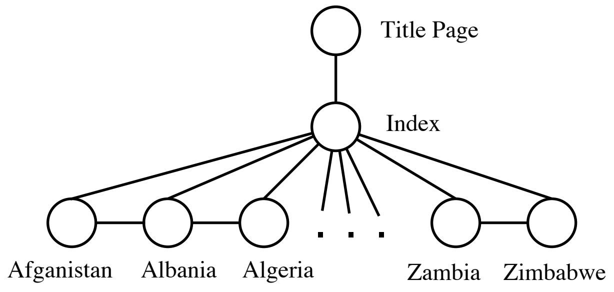

Fig. 10: Structure of the Sample Hypermedia Document

Hypercard was chosen because of its availability, its ability to interface with external routines, and its resources to quickly create interfaces. While the information gathering, and visualization are specific to the hypermedia system being used, the genetic algorithm and neural network clustering routines are generic.

We tested our algorithm by clustering the maps of the World Fact Book. This 1.2 megabyte hypermedia document contains digitized maps of 247 countries. An index card provides direct access to each map, and the maps are linked alphabetically as shown in Figure 10. This document is typical of educational materials being offered on CD-ROM and laserdisc (e.g. images of artwork) where there is a large amount of graphical information with little or no structure. There are no keywords for the concept based clustering, and not enough links for the structure based clustering. The user may choose to look at countries based on their geography, economy, political system, or the countries the user visited on vacation last year. The information is generic, with each user giving it unique meaning.

Five paths were taken through the stack, each stored in its own list:

1. NORTH AMERICA List: Title → Index → United States → Index → Canada → Index → Mexico → Index
2. VACATION List: Title → Index → United States → Index → Canada → Index → Mexico → Index → American Samoa → Index → U. K. → Index
3. 'A' COUNTRIES List: Title → Index → Afganistan → Albania → Algeria → American Samoa → Andorra → Angola → Index
4. ASIA AND EUROPE List: Title → Index → China → Index → Taiwan → Index → Vietnam → Index → U. K. → Index → Germany → Index
5. EUROPE AND ASIA List: Title → Index → U. K. → Index → France → Index → Spain → Index → China → Index → Taiwan → Index

The genetic algorithm takes 25 seconds to return the clusters shown in Figure 11. The neural network takes 30 seconds to return the clusters shown in Figure 12.

The genetic algorithm and neural network create similar clusterings of the hypermedia document's nodes. The genetic algorithm generates a strict hierarchy with an arbitrary number of levels where the relationships between the clusters are shown by their relative positions in the hierarchy. The neural network generates only two levels of clustering but the overview shows more subtle relationships between the clusters. The genetic algorithm generates deeper, more structured indices than the neural network. The neural network gives a better overall picture of how clusters are related.

The neural network clustering can help a new user see a global picture of the hypermedia document. With the neural network clustering, nodes used most generally (such as the title screen, or the table of contents) will be found near the center of the array. Nodes with more specific uses will be found towards the corners. This can be seen in Figure 12 where the nodes used in all 5 lists (Title, Index) ended up in the center of the array. The nodes from the 'Asia and Europe' and

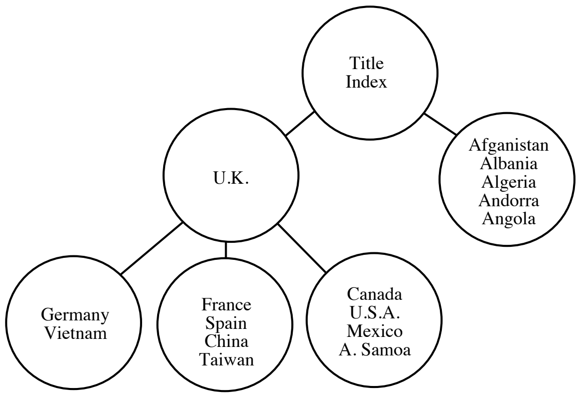
Title Index
U.K.
Afganistan Albania Algeria Andorra Angola
Germany Vietnam
France Spain China Taiwan
Canada U.S.A. Mexico A. Samoa

Fig. 11: Clusters produced by the Genetic Algorithm

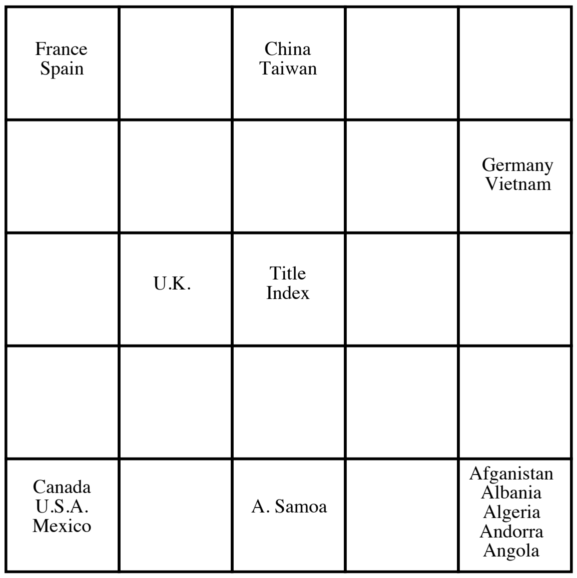

Fig. 12: Clusters produced by the Neural Network

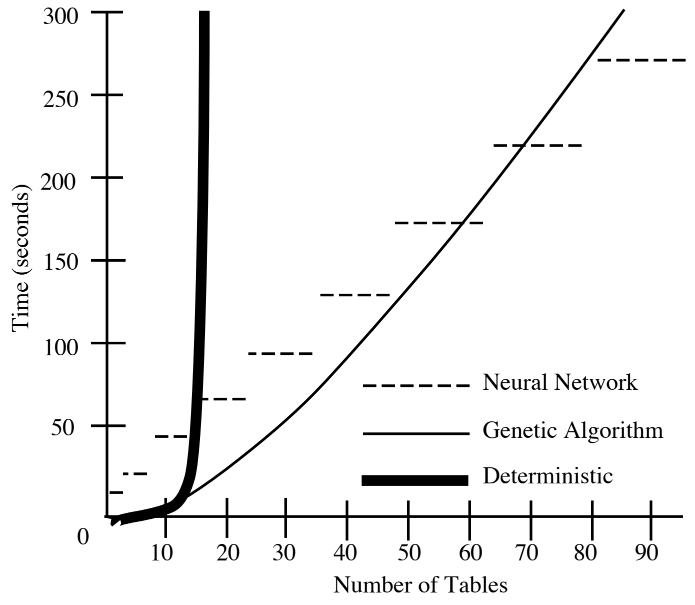

Fig. 13: Time Comparison of Different Clustering Algorithms

'Europe and Asia' lists take up the top of the array with the common 'China' and 'Taiwan' nodes in the middle. The nodes from the 'Vacation' list take up the lower left corner with the 'American Samoa' node midway between the 'Vacation' group and the 'A Countries' group, as it belongs to both.

The genetic algorithm clustering can help a user familiar with the system categorize clusters into a hierarchical relationship. General concepts are found at the top of the hierarchy (such as the title screen, or the table of contents) and more specific concepts are found at the bottom of the hierarchy, near the actual nodes being clustered.

The two algorithms have comparable execution times. With inputs of less than 14 nodes our genetic algorithm, on average, found a clustering better than 99% of the possible clusterings. It was impractical to do comparisons with more than 13 input nodes due to the running time of the deterministic program. For a comparison of running times of the deterministic, genetic algorithm, and neural network clustering algorithms see Figure 13. The time for running the neural network is directly related to the size of the array. As the array size is based on the $\sqrt{number\ of\ nodes}$, the time complexity of the neural network is a step function.

Clustering will be an ongoing process in the hypermedia document. New users will begin using the hypermedia document, current users will return to search for new information. Perhaps even new nodes will be added. More and more relationships will connect the nodes together, and more nodes will be brought into the clustering. These new relationships may be very different from the older relationships so the system should dynamically support both the new and current users.

Totally re-clustering the nodes will allow the index to show the most effective clustering based on all of the users' previous experiences. This will be very useful to new users, but may be confusing to current users when the clustering patterns change, possibly dramatically. Instead the clustering can be performed incrementally. The current clusters can be used as an input to both the genetic algorithm and the neural network, instead of starting the clusters from scratch. This will allow the clusters to change slowly, giving the current users a common framework and adding in each new user's contributions.

More than one set of indices can be maintained in the system. New users can start with the most up-to-date indexing information, while current users can access the hypermedia document using their familiar clusters. However the clusters are maintained, the underlying hypermedia document remains untouched.

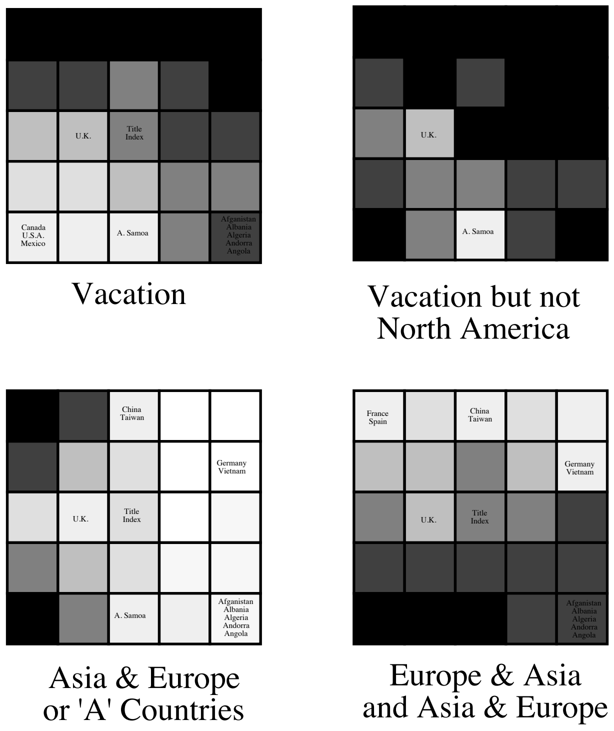

Fig. 14: Sample Index Queries using Neural Network

7. USES

The clusters that have been generated can be used for several purposes. They can aid both the user and the owner of the hypermedia document.

When a user wishes to start a new query, they can look through the existing set of topics that have been clustered. One of these topics may match up with their needs giving them an excellent starting point for their searchess. They may find that their area of interest straddles two existing queries. Using the existing lists the user can reduce their search time in the hypermedia document as shown in Figures 14 and 15.

Figure 14 shows how the clusters in Figure 12 can be highlighted. Each of the arrays in Figure 14 is in the same orientation as the larger (and more readable) Figure 12. The user can choose to see the tables important to someone interested in different concepts.

By selecting the appropriate concepts the clusters highlight the appropriate areas of the hypermedia document. Important areas are shown in white; unimportant areas are shown in black. The brightness of the region shows how relevant the nodes in that region are to the suggested concept areas. Since the neural network forms a topological map of the input lists, these input lists allow us to find the literal peaks and valleys of interest. The user now has quick access to sets of nodes without having to browse through the existing hypermedia document structure.

As well as highlighting the important concepts, the clustering diagram also shows relationships between the highlighted concepts, and how general or specific the highlighted concepts are by their location in the array. Thus it gives much more information to the user than simply highlighting concepts based on the usage percentages directly, or using a general overview diagram of the hypermedia document.

It is unlikely that any of the existing concepts will exactly match the needs of a new user so the ability to find 'nearby' nodes is very important. This can only be provided by integrating the experiences of many users. This information can be especially important if the user needs to find intermediate nodes that link together nodes of interest. The ability to see 'nearby' nodes is also important if the user has an inexact concept. The existing concept lists can give the user a good

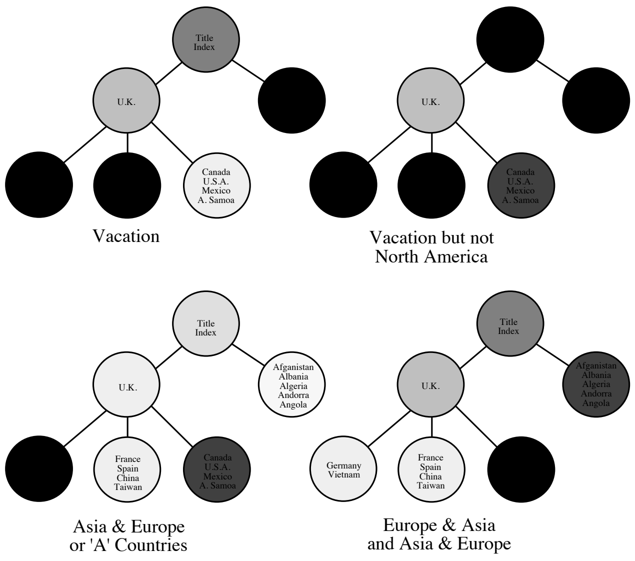
Title Index
U.K.
Canada U.S.A. Mexico A. Samoa
Vacation
U.K.
Canada U.S.A. Mexico A. Samoa
Vacation but not North America
Title Index
U.K.
Afganistan Albania Algeria Andorra Angola
France Spain China Taiwan
Canada U.S.A. Mexico A. Samoa
Asia & Europe or 'A' Countries
Title Index
U.K.
Afganistan Albania Algeria Andorra Angola
Germany Vietnam
France Spain China Taiwan
Europe & Asia and Asia & Europe

Fig. 15: Sample Index Query using Genetic Algorithm

starting point to begin browsing through the nearby nodes to see if they are of interest.

Figure 15 shows how the clusters in Figure 11 can be highlighted. As with the neural network version, the user can choose combinations of the existing concepts to see the nodes that are of personal importance.

Using the clustering information provided by the users, the owner of the hypermedia document can find which nodes are in common usage among which groups. For example, all the users would require access to the most commonly used tables but perhaps only certain groups are accessing certain tables. Using the clusters, the owner can determine which nodes a person will likely need based on their areas of interest, because those are the nodes that people with similar interests have used in the past.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed, implemented, and compared two adaptive clustering algorithms for hypermedia documents. These algorithms hierarchically cluster nodes according to the user's use of the hypermedia document, rather than the author's intended uses. These algorithms can cluster hypermedia documents where keywords do not exist (including hypermedia documents containing foreign languages, movies, and pictures), or where the link structure is too general. This hierarchy of clusters gives each user a personal index into the hypermedia document.

Both the genetic algorithm and the neural network create clusters within a reasonable amount of time. The genetic algorithm generates a strict multi-level hierarchy of clusters. The neural network generates a two dimensional map of the clusters. Both algorithms allow the user to view the hypermedia document in a personalized way.

We are currently enhancing our implementation in the following ways: 1. Allowing the algorithms to reflect incremental updates to the structure of the hypermedia document, including the addition and deletion of nodes, without recomputing all the clusters, 2. Using available keywords to name the clusters that have been created, 3. Clustering over multiple hypermedia documents and multiple sites to combine nodes from multiple documents into the same cluster, 4. Applying this approach to other database systems, and considering alternative clustering techniques.

Acknowledgements — We would like to thank the reviewers for their valuable comments.

REFERENCES

[1] R. Botafogo and B. Schneiderman. Identifying aggregates in hypertext structures. In *Proceedings of the Hypertext '91 Conference*, pp. 63–74 (1991).

[2] G. Boy. Indexing hypertext documents in context. In *Proceedings of the Hypertext '91 Conference*, pp. 51–62 (1991).

[3] V. Bush. As we may think. *The Atlantic*, pp. 101–108 (1945).

[4] M. Casanova, L. Tucherman, M. Lima, J. Netto, N. Rodriguez, and L. Soares. The nested context model for hyperdocuments. In *Proceedings of the Hypertext '91 Conference*, pp. 193–202, San Antonio, Texas (1991).

[5] W. Croft and H. Turtle. A retrieval model incorporating hypertext links. In *Proceedings of the Hypertext '89 Conference*, pp. 213–224, Pittsburg, Penn. (1989).

[6] D. Crouch, C. Crouch, and G. Andreas. The use of cluster hierarchies in hypertext information retrieval. In *Proceedings of the Hypertext '89 Conference*, pp. 225–238, Pittsburg, Penn. (1989).

[7] M. Frisse and S. Cousins. Information retrieval from hypertext: Update on the dynamic medical handbook project. In *Proceedings of the Hypertext '89 Conference*, pp. 199–212, Pittsburg, Penn. (1989).

[8] P. Garg. Abstraction mechanisms in hypertext. *Communications of the ACM*, **31**(7):862–870 (1988).

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine learning.* Addison-Wesley, New York (1989).

[10] F. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, **31**(7):836–852 (1988).

[11] Y. Hara, A. Keller, and G. Wiederhold. Implementing hypertext database relations through aggregations and exceptions. In *Proceedings of the Hypertext '91 Conference*, pp. 75–90 (1991).

[12] A. Johnson and F. Fotouhi. Automatic touring in a hypertext system. In *Proceedings of the 12th IEEE International Phoenix Conference on Computers and Communication*, pp. 524–530, Tempe, Arizona (1993).

[13] A. Johnson and F. Fotouhi. Adaptive indexing in very large databases. *Journal of Database Management*, **6**(1):4–12 (1995).

[14] A. Johnson, F. Fotouhi, and N. Goel. Adaptive clustering of scientific data. In *Proceedings of the 13th IEEE International Phoenix Conference on Computers and Communication*, pp. 241–247, Tempe, Arizona (1994).

[15] T. Kohonen. The self-organizing feature map. *Proc. IEEE*, **78**(9):1464–1480 (1990).

[16] J. Nielsen. *Hypertext and Hypermedia.* Academic Press, San Diego (1990).

[17] R. Nielsen. *Neurocomputing.* Addison-Wesley, New York (1990).

[18] I. Tomek, S. Khan, T. Muldner, M. Nassar, G. Novak, and P. Proszynski. Hypermedia - introduction and survey. *Journal of Microcomputer Applications*, **14**:63–103 (1991).