Learning Implicit User Interest Hierarchy for Context in Personalization

By Hyoung R. Kim and Philip K. Chan
Florida Institute of Technology Technical Report CS-2002-15
hokim@fit.edu, pkc@cs.fit.edu

ABSTRACT
To provide a more robust context for personalization, we desire to extract a continuum of general (long-term) to specific (short-term) interests of a user. Our proposed approach is to learn a user interest hierarchy (UIH) from a set of web pages visited by a user. We devise a divisive hierarchical clustering (DHC) algorithm to group words (topics) into a hierarchy where more general interests are represented by a larger set of words. Each web page can then be assigned to nodes in the hierarchy for further processing in learning and predicting interests. This approach is analogous to building a subject taxonomy for a library catalog system and assigning books to the taxonomy. Our approach does not need user involvement and learns the UIH "implicitly." Furthermore, it allows the original objects, web pages, to be assigned to multiple topics (nodes in the hierarchy). In this paper, we focus on learning the UIH from a set of visited pages. We propose a few similarity functions and dynamic threshold-finding methods, and evaluate the resulting hierarchies according to their meaningfulness and shape.

Keywords
user interest hierarchy, user profile, clustering algorithm, concept clustering

1. INTRODUCTION

When a user browses the web, at different times, she could be accessing pages pertaining to different topics. For example, she might be looking for research papers at one time and airfare information for conference travel at another. That is, a user can exhibit different kinds of interests at different times, which provide different contexts underlying a user's behavior. However, different kinds of interests might be motivated by the same kind of interest at a higher abstraction level (computer science research, for example). That is, a user might possess interests at different abstraction levels—the higher-level interests are more general, while the lower-level ones are more specific. Furthermore, more general interests, in some sense, correspond to longer-term interests, while more specific interests correspond to shorter-term interests. During a browsing session, the general interests are back on one's mind, while specific interests are one's current foci. Unlike News Dude [2], which generates a long-term and a short-term model, we model a continuum of long-term to short-term interests. We believe identifying the appropriate context underlying a user's behavior is important in more accurately pinpointing her interests. The web is not static — new documents and new words/phrases are created every day. Most clustering methods assume each object (document) is represented by a fixed number of features (words/phrases). This representation is inadequate in a dynamic environment like the web. Consider how a librarian would form taxonomy of the subjects for all the books in the library. She would first identify the subject(s) of a book and then cluster all the books on the bases of the subject. Finally, books are categorized by the taxonomy. STC [14] does not rely on a fixed vector of word features in clustering documents. We use a similar approach—instead of clustering documents, we cluster features in the documents; documents are then assigned to the clusters. We propose to model general/long-term and specific/short-term interests with a concept hierarchy called *User Interest Hierarchy* (UIH). The resulting hierarchy (UIH) is used to build *Page Interest Estimator* (PIE)'s [3] as well as provides a context. For each cluster in UIH, the associated documents are used as positive examples for learning a PIE. The constructed UIH and its corresponding learned PIE's are used for estimating interest of a new document. However, current clustering methods do not generate clusters that possess all of the key characteristics we desire in a UIH.

The most common and obvious solution for building a UIH is for the user to specify their interests *explicitly*. However, the *explicit* approach includes these disadvantages:

- Time and effort in specifying her interests.
- User's interest may change over time.

Alternatively, an *implicit* approach can identify a user's interests by inference.

The main objective of this research is to build UIH's without the user's involvement (*implicitly*). We devise a divisive hierarchical clustering (DHC) algorithm that constructs such a hierarchy and supports overlapping clusters of the original objects (web pages in our case). We believe our approach has significant benefits and possesses interesting challenges. The main contributions of this work are the characterization of a user interest hierarchy, an algorithm that constructs a UIH, similarity functions, dynamic threshold-finding methods, and evaluation of our techniques based on real data collected from our departmental web server.

The rest of this paper is as follows: Section 2 discusses related work in clustering algorithms and building user interest profiles; Section 3 introduces user interest hierarchies (UIH's); Section 4 details our approach towards building implicit UIH's; Section 5 describes our experiments; Section 6 analyzes the results from the experiments; Section 7 summarizes our findings and suggests possible future work.

2. RELATED RESEARCH

Agglomerative (bottom-up) hierarchical clustering algorithms initially put every object in its own cluster and then repeatedly merge similar clusters together, resulting in a tree shape structure that contains clustering information on many different levels [12]. Merges are usually binary—merging two entities, which could be clusters or initial data points. Hence, each parent is forced to have two children in the hierarchy. Divisive (top-down) hierarchical clustering algorithms are similar to agglomerative ones, except that initially all objects start in one cluster which is repeatedly split. Splits are usually binary and one usual stopping criterion is the desired number of clusters [4]. Our divisive algorithm does not necessarily generate binary splits and uses a minimum cluster size as one of the stopping criteria. Partitioning clustering algorithms such as the *K*-means algorithm initially create a partitioning of *K* clusters. Those initial *K* clusters are then iteratively refined to achieve the final clustering of *K* clusters. A major drawback of this approach is that the number of clusters must be specified beforehand as an input parameter, however Perkowitz developed a method to automatically determine the value of *K*. They ran the *K*-means algorithm multiple times, starting with a large value and gradually decreasing it. They were able to efficiently determine a good value for *K* during the clustering of web pages [9]. Our algorithm only needs to cluster strongly connected words, but the *K*-means algorithm divides whole words into *K* clusters without removing weak relations. COBWEB is a incremental conceptual clustering algorithm. Each cluster records the probability of each attribute and value, and the probabilities are updated every time an object is added [4]. However, instead of using *category utility* to determine if child clusters are generated, we use a graph-based method and a different similarity function.

To build user interest profiles that can be used for web personalization, Richardson and Domingos [10] enhanced PageRank by using a more intelligent web-surfer. This method collects relevant web pages based only on queries, probabilistically combined page contents, and link structure. Research has also been performed on a method to group web pages into distinct topics and to list the most authoritative/informative web pages in each topic. The similarity metric that is used incorporates comprehensive information regarding text, hyperlink structure, co-citation, and the unsupervised clustering method based on spectral graph partitioning using normalized cut [7]. Our method is only concerned with the text but allows overlapping clusters. A news agent called News Dude, developed by Billsus and Pazzani [2], learns which stories in the news a user is interested in. The news agent uses a multi-strategy machine learning approach to create separate models of a user's short-term and long-term interests. Unlike News Dude, in out approach we model a continuum of long-term to short-term interests. Syskill & Webert [8] use a predefined profile, which significantly increases the classification accuracy on previously unseen web pages. They emphasize the importance of a user profile. Perkowitz and Etzioni [9] introduced SCML, a concept learning algorithm that only extracts some concepts in a set of data.

3. USER INTEREST HIEREARCHY

A user interest hierarchy (UIH) organizes a user's general to specific interests. Towards the root of a UIH, more general (longer-term) interests are represented by larger clusters of words while towards the leaves, more specific (shorter-term) interests are represented by smaller clusters of words. To generate a UIH for a user, our clustering algorithm (details in Sec. 4) accepts a set of web pages visited by the user as input. We use only the words in a web page and ignore link or image information. The web pages are stemmed and filtered by ignoring the most common words listed in a stop list [5]. Table 1 has a sample data set.

Page	Content
1	ai machine learning ann perceptron
2	ai machine learning ann perceptron
3	ai machine learning decision tree id3 c4.5
4	ai machine learning decision tree id3 c4.5
5	ai machine learning decision tree hypothesis space
6	ai machine learning decision tree hypothesis space
7	ai searching algorithm bfs
8	ai searching algorithm dfs
9	ai searching algorithm constraint reasoning forward checking
10	ai searching algorithm constraint reasoning forward checking

Table 1: Sample data set

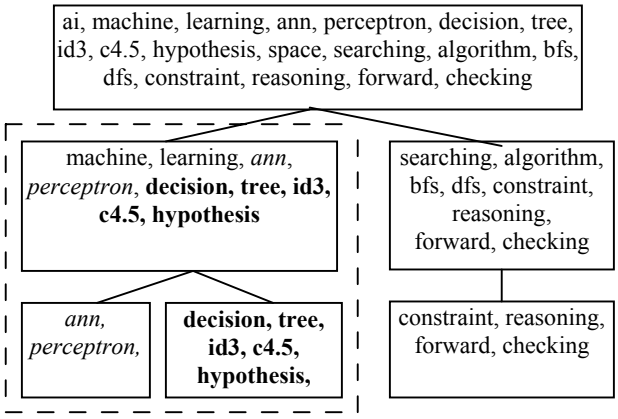

Figure 1: Sample user interest hierarchy

Numbers in the left represent individual web pages; content has words stemmed and filtered through stop list. These words in the web pages can be represented by a UIH as shown in Figure 1. Each cluster node can represent a conceptual relationship, for example 'perceptron' and 'ann' (in *italic*) can be categorized as belonging to neural network algorithms, whereas 'id3' and 'c4.5' (in **bold**) in another node cannot. Words in these two nodes are mutually related to some other words such as 'machine' and 'learning'. This set of mutual words, 'machine' and 'learning', performs the role of connecting *italic* and **bold** words in sibling clusters and forms the parent cluster. We illustrate this notion in the dashed box in Figure 1.

4. APPROACH

We desire to cluster web pages to provide contexts for predicting user interest. To allow overlapping clusters of pages, instead of clustering pages directly, we cluster the words in pages and pages are assigned to clusters subsequently. That is, instead of directly clustering the original objects (web pages), we first cluster the features (words) of the objects and then the objects are assigned to the clusters based on the features in each cluster. During clustering, the similarity/distance between features are based on their relationship with the objects. Consequently, objects are clustered based on possessing similar related features and each object may belong to multiple clusters. Since the more challenging step is the initial hierarchical clustering of the features, our primary focus for this paper is on devising and evaluating algorithms for this step.

Our divisive hierarchical clustering (DHC) algorithm recursively partitions the words into smaller clusters, which represent more related words. We assume words occurring close to each other (within a window size) are related to each other. We investigated a few similarity functions that measure how close two words are related. We also investigated techniques that dynamically locate a threshold that decides whether two words are strongly related or not. If two words are determined to be strongly related, they will be in the same cluster; otherwise, they will be in different clusters. In the following subsections, we detail our algorithm, similarity functions, threshold-finding techniques, choice of window size, and minimum cluster size for leaves.

4.1 Algorithm

Our algorithm is a divisive hierarchical clustering method called DHC, that recursively divides clusters into child clusters until it meets the stopping conditions. Figure 2 illustrates the pseudo code for the HDC algorithm. In preparation for our clustering algorithm, we extract words from web pages visited by the user, filter them through a stop list, and stem them [5]. Using a similarity function, we calculate the strength of the relationship between a pair of words. We then build a weighted undirected graph with each vertex representing a word and each weight denoting the similarity between two words. Since related words are more likely to appear in the same document than unrelated terms, we measure co-occurrence of words in a document. Given the graph, called SimilarityMatrix, the clustering algorithm recursively partitions the graph into subgraphs, called Cluster, each of which represents a sibling node in the resulting UIH. Documents that contain words in a cluster are in the cluster. Note that a document can have terms in different clusters, hence, a document can be in more than one cluster. At each partitioning step, edges with "weak" weights are removed and the resulting connected components constitute sibling clusters (we can also consider cliques as clusters, but more computation is required). We treat determining what value is considered to be "strong" or "weak" as another clustering problem (more details in Sec. 4.3). The recursive partitioning process stops when one of the stopping criteria is satisfied. The first criterion is when the current graph does not have any connected components after weak edges are removed. The second criterion is a new child cluster is not formed if the number of words in the cluster falls below a predetermined threshold.

The CalculateSimilarityMatrix function takes a similarity function (details in Sec. 4.2), cluster, and window size as parameters and return the similarity matrix, where the window size affects how far two words (in terms of number of words) can be to be considered as related. The CalculateThreshold function takes a threshold-finding method and similarity matrix as parameters and returns the threshold. The similarity function and threshold-finding method greatly influence the clustering algorithm and are discussed next.

UIH (Examples, SIMILARITYFUNCTION, FINDTHRESHOLD, WindowSize)
Examples: A set of web pages visited by the user.
SIMILARITYFUNCTION: A function that calculates the "closeness" of two words.
FINDTHRESHOLD: A function that calculates the cutoff value for determining strong and weak similarity values.

WindowSize: the maximum distance (in number of words) between two related words in calculating their similarity value.

1. Words are extracted from Examples, stemmed, and filtered through a stop list.
2. Cluster ← distinct words [with information of web page membership]
3. Return **DHC**(Cluster, SIMILARITYFUNCTION, FINDTHRESHOLD, WindowSize)

DHC (Cluster)
1. SimilarityMatrix ← CalculateSimilarityMatrix (SIMILARITYFUNCTION, Cluster, WindowSize)
2. Threshold ← CalculateThreshold(FINDTHRESHOLD, SimilarityMatrix)
3. If all similarity values are the same or a threshold is not found
   Return EmptyHierarchy
4. Remove weights that are less than Threshold from SimilarityMatrix
5. While (ChildCluster←NextConnectedComponent (SimiarityMatrix))
   If size of ChildCluster >= MinClusterSize
   ClusterHierarchy←ClusterHierarchy + ChildCluster + **DHC**(ChildCluster, SIMILARITYFUNCTION, FINDTHRESHOLD, WindowSize)
6. Return ClusterHierarchy

Figure 2: DHC algorithm

4.2 Similarity Functions

The similarity function calculates how strongly two words are related. Since related words are likely to be close to each other than unrelated words, we assume two words co-occurring within a window size are related. To simplify our discussion, we have been assuming the window size to be the entire length of a document (details in Sec. 4.4). That is, two words co-occur if they are in the same document.

4.2.1 AEMI

We use *AEMI* (Augmented Expected Mutual Information) [3] as a similarity function. *AEMI* is enhanced version of *MI* (Mutual Information) and *EMI* (Expected Mutual Information). Unlike *MI* which considers only one corner of the confusion matrix and *EMI* which sums the *MI* of all four corners of the confusion matrix, *AEMI* sums supporting evidence and subtracts counter-evidence. Chan [3] demonstrates that *AEMI* could find more meaningful multi-word phrases than *MI* or *EMI*. Concretely, consider *A* and *B* in *AEMI*(*A*,*B*) are the events for the two words. $P(A=a)$ is the probability of a document containing *a* and $P(A=\bar{a})$ is the probability of a document not having term *a*. $P(B=b)$ and $P(B=\bar{b})$ are defined likewise. $P(A=a,B=b)$ is the probability of a document containing both terms *a* and *b*. These probabilities are estimated from the documents visited by the user. *AEMI(A,B)* is defined as:

$$AEMI(A,B) = P(a,b)\log\frac{P(a,b)}{P(a)P(b)} - \sum_{(A=a,B=\bar{b})(A=\bar{a},B=b)} P(A,B)\log\frac{P(A,B)}{P(A)P(B)}$$

The first term computes supporting evidence that *a* and *b* are related and the second term calculates counter-evidence.

Using our running example in Figure 1, Table 2 shows a few examples of how *AEMI* is computed. The *AEMI* value between '*searching*' and '*algorithm*' is 0.36, which is higher than the *AEMI* value between '*space*' and '*constraint*', –0.09.

$P(a)$	$P(\bar{a})$	$P(b)$	$P(\bar{b})$	$P(ab)$	$P(\bar{a}b)$	$P(a\bar{b})$	$AEMI(a,b)$
a = searching, b = algorithm							
0.4	0.6	0.4	0.6	0.4	0	0	0.36
a = space, b = constraint							
0.2	0.8	0.2	0.8	0	0.2	0.6	-0.09
a = ann, b = perceptron							
0.2	0.8	0.2	0.8	0.2	0	0	0.32

Table 2: *AEMI* values

4.2.2 AEMI-SP

Inspired by work in the information retrieval community, we would like to enhance *AEMI* by incorporating a component for inverse document frequency (IDF) in the similarity function. The document frequency of a word calculates the number of documents that contain the word. Words that are commonly used in many documents are usually not informative in characterizing the content of the documents. Hence, the inverse document frequency (the reciprocal of document frequency) measures how informative a word is in characterizing the content. Since our formulation is more sophisticated than IDF and it involves a pair of words rather than one word in IDF, we use a different name and call our function *specificity* (*SP*).

We estimate the probability of word occurrence in documents instead of just document frequency so that we can scale the quantity between 0 and 1. We desire to give high SP values to words with probability below 0.3 (approximately), gradually decreasing values from 0.3 to 0.7, and low values above 0.7. This behavior can be approximated by a sigmoid function, commonly used as a smoother threshold function in neural networks, though ours needs to be smoother. Figure 3 shows the shape of the *SP* function with respect to *m*, where *m* is defined as: *MAX (P(a), P(b))*. We choose the larger probability so that *SP* is more conservative. *SP(m)* is defined as:

$$1/(1 + exp(0.6 \times (m \times 10.5 - 5))),$$

where the factor 0.6 smoothes the curve, and constants 10.5 and –5 shift the range of m from between 0 and 1 to between -5 and 5.5. The new range of -5 and 5.5 is slightly asymmetrical because we would like to give a small bias to more specific words. For instance, for *a* = 'ann' and *b* = 'perceptron', m is 0.2 and *SP(m)* is 0.85, but for ='machin' and *b*='ann', *m* is 0.6 and *SP(m)* is 0.31.

Our similarity function *AEMI-SP* is defined as: *AEMI* * SP/2. The usual range for *AEMI* is 0.1 – 0.45 and SP is 0 – 1. To scale SP to a similar range as *AEMI,* we divide SP by

2. For example in Table 4 the *AEMI*-SP value for 'searching' and 'algorithm' is lower than the value for 'ann'' and 'perceptron' because the SP value for 'ann' and 'percetpron' is higher even though the *AEMI* value is lower.

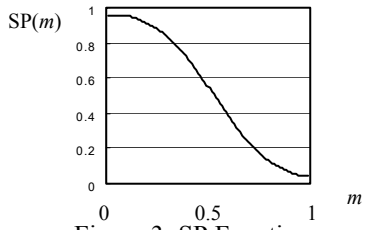

Figure 3: SP Function

	AEMI	*SP*	*AEMI-SP*
a = searching *b = algorithm*	0.36	0.62	0.113
a = ann *b = perceptron*	0.32	0.85	0.137

Table 4: *AEMI-SP* values

4.2.3 Other Similarity Functions

We also investigated other existing similarity functions. The *Jaccard* function [6] is defined as: $\frac{P(a,b)}{P(a \cup b)}$. Although *Jaccard* produces meaningful clusters, it did not generate suitable hierarchical clusters. When we calculated the similarity matrix on the sample data using the *Jaccard* function, the related value of '*ai*' was expected to be very small, since the words were very general; however, the computed *Jaccard* values was bigger than average, which made it hard to make child clusters, which means it is not proper for making hierarchical clusters.

For instance, using our running example in Figure 1, the *Jaccard* value between 'ai' and 'machine' is 0.6 and the value between 'ai' and 'search' is 0.5. If the threshold is 0.49, both pairs are in the same cluster and 'ai' may perform the role to connect 'machine' and 'search'. Even though if the threshold is 0.55, 'ai' still remains in the child cluster with 'machine' (since their similarity value is over the threshold), which is a wrong decision.

The *MIN* method is defined as *MIN*(*P*(*a*|*b*), *P*(*b*|*a*)). The idea is that if we assign the same similarity value to connected words and connecting words, they would go together. For instance in Figure 1, 'ai' connects 'machine' and 'searching', so they were grouped together in one cluster. However, when they were divided into child clusters, 'ai' should be removed because 'ai' is too general. But *MIN* (*P*('ai'|'machine'), *P*('machine'|'ai')) still yielded relatively higher value than the average. Alternatively, the *MAX* function defined as *MAX* (*P*(*a*|*b*), *P*(*b*|*a*)) did not distinguish the value for 'ai' and 'machine' and the value for 'machine' and 'learning', even though the latter pair has a much stronger relationship. Since *Jaccard, MIN,* and *MAX* did not generate desirable cluster hierarchies, we excluded them from further experiments.

4.3 Threshold-finding Methods

Instead of using a fixed user-provided threshold (as in STC [14]) to differentiate strong from weak similarity values between a pair of words, we examine methods that dynamically determine a reasonable threshold value. Weights with weak similarity are removed from SimilarityMatrix and child clusters are identified (Sec. 3).

4.3.1 Valley

To determine the threshold, we would like to find a sparse region that does not have a lot of similar values. That is, the frequency of weights in that region is low. We first determine the highest observed and lowest desirable similarity values and quantize the interval into ten regions of equal width. The lowest desirable similarity value is defined as the value achieved by a pair of words that occur together only in one document. We then determine the frequency of values in each region.

Generally, lower weights have a higher frequency and higher weights have a lower frequency. If the frequency monotonically decreases with regions of higher weights, picking the region with the lowest frequency will always be the region with the highest weights. Unfortunately, the threshold will be too high and too many edges will be cut. In this case the threshold is set to be the average plus one standard deviation (biasing to removing more edges with lower weights).

However, if the frequency does not decrease monotonically, we attempt to identify the "widest and steepest" valley. Steepness can be measured by the slopes of the two sides of a valley and the width of how many regions the valley covers. Since the regions are of equal width, we calculate the "quality" of a valley by: $\sum_{i,j} \left| freq_i - freq_j \right|$, where i and j are successive regions on the two sides of a valley. Once the widest and steepest valley is located, we identify the threshold in the region that constitutes the bottom (lowest frequency) of the valley. For example, in Table 5, there are three valleys: one from Region 0 through 3, (quality is 17), another one from Region 3 through 5, (quality is 14), and the last one is from Region 5 through 9, (quality is 15). Therefore, the widest and steepest valley is the first valley and its bottom is in Regions 1 and 2.

To identify the threshold inside the bottom region, we ignore the frequency information and find two clusters of similarity values. In this case, it is a one-dimensional two-cluster task, which can be accomplished by sorting the weights and splitting at the largest gap between two successive weights (LargestGap in Sec. 4.4). In our example in Table 5, since the bottom has zero frequency, any value between .28 and .30 can be the threshold.

Region	Range	Freq.	# of Children
0	0.27 <= x < 0.28	16	Not counted
1	0.28 <= x < 0.29	0	Not counted
2	0.29 <= x < 0.30	0	Not counted
3	0.30 <= x < 0.31	1	Not counted
4	0.31 <= x < 0.32	0	Not counted
5	0.32 <= x < 0.33	13	6
6	0.33 <= x < 0.34	0	1
7	0.34 <= x < 0.35	0	1
8	0.35 <= x < 0.36	0	1
9	0.36 <= x	2	Not applicable

Table 5: Distribution of frequency and number of children

4.3.2 MaxChildren

The MaxChildren method selects a threshold such that maximum of child clusters are generated. This ensures that the resulting hierarchy does not degenerate to a tall and thin tree (which might be the case for other methods). This preference stems from the fact that topics are in general more diverse than detailed and the library catalog taxonomy is typically short and wide. MaxChildren calculates the number of child clusters for each boundary value between two quantized regions. To guarantee the selected threshold is not too low, the method ignores the first half of the boundary values. For example, in Table 5, the boundary value 0.33 (between Regions 5 and 6) generates the most children and is selected as the threshold.

4.3.3 Other Threshold-finding Methods

There are some other threshold-finding methods that we initially studied but are inferior to Valley or MaxChildren and are not included in this paper. LargestGap sorts the values and split at the largest gap between two successive values (the same method used in the Valley method *after* the bottom of the largest valley is found). Again this is motivated by trying to form two clusters in a one-dimensional space. However, in our initial experiments, the largest gap is close to the largest observed value and hence the resulting tree is usually too small. To prevent the threshold being too large, Top30% selects a threshold that retains values in the top 30%. However, this method generates tall and thin trees. To keep 'abnormally' large values, we also studied Average+StandardDeviation to select a threshold a standard deviation larger than the average. This is later combined into the Valley method.

4.4 Window Size and Minimum Size of a Cluster

The window size parameter specifies the maximum 'physical' distance (in terms of number of words) between a pair of words for consideration of co-occurrence. We have been using the entire document length as the window size to simplify our discussion. However, considering two words occurring in the same page as related might be too optimistic. Hence, we investigated smaller window sizes that roughly cover a paragraph (e.g., 100 words) or a sentence (e.g., 15 words). However, in our experiments the window size does not make a significant difference.

The minimum size of a cluster affects the number of clusters. A larger number of clusters makes the hierarchy less comprehensible and requires more computation. We picked 4 as the minimum size of a cluster.

5. EXPERIMENTS

Experiments were conducted on data obtained from our departmental web server. By analyzing the server access log from January to April 1999, we identified hosts that were accessed at least 50 times in the first two months and also in the next two months. We filtered out proxy, crawler, and our computer lab hosts, and identified "single-user" hosts, which are at dormitory rooms and a local company [3]. We yielded 13 different users and collected the web pages they visited. The number of words on the web pages visited by each user was on the average 1,918, minimum number of words was 340, and maximum was 3,708. We evaluate the effectiveness of our algorithms by analyzing the generated hierarchies in terms of *meaningfulness* and *shape*. Separate experiments were conducted to evaluate the effectiveness of different similarity functions, threshold-finding methods, and window sizes.

6. ANALYSIS

To evaluate a UIH, we use both qualitative and quantitative measures. Qualitatively, we examine if the cluster hierarchies are reasonably describing some topics (*meaningfulness*). Quantitatively, we measure the *shape* of the cluster trees by calculating the average branching factor [11] (ABF). ABF is defined as the total number of branches of all non-leaf nodes divided by the number of non-leaf nodes.

We categorized meaningfulness as 'good', 'fair,' or 'bad'. Since the leaf clusters should have specific meaning and non-leaf clusters are hard to interpret due to their size, we only evaluated the leaf clusters for meaningfulness. This measure is based on interpretability and usability [6] and checks two properties of the leaf the existence of related words and possibility of combining words. For instance for the related words, consider 'formal', 'compil', 'befor', 'graphic', 'mathemat', and 'taken' are in a cluster, even though 'befor' and 'taken' do not have any relationship with other words, since other words are classified as a class name, this cluster is evaluated as 'good'. And for the possibility of combining words, consider 'research', 'activ', 'class', and 'web' are in a cluster. In this case the meaning of the cluster can be estimated as 'research activity' or 'research class', so we regard this cluster as good [13]. A cluster is marked as 'good' when it has more than 2/5 of the words that are related or has more than 2 possible composite phrases. This is hard to measure, so we tried to be skeptical as much as possible. For example, suppose a cluster has 'test', 'info', 'thursdai', 'pleas', 'cours', 'avail', and 'appear'. In this case one can say 'test

info' or 'cours info' are possible composite phrases, but 'test info' does not have any conceptual meaning in our opinion, so we did not count that phrase. A cluster is marked as 'bad' when a leaf cluster has more than 15 words because a big leaf cluster is hard to interpret. 'Fair' leaf clusters are those that are neither good nor bad.

We categorized shape as 'thin', 'medium,' or 'fat'. If a tree's ABF value is 1, the tree is considered a 'thin' tree (marked as 'T' in the following tables). If the ABF value of a tress is at least 10, the tree is considered a 'fat' tree (marked as 'F'). The rest are 'medium' trees (marked as 'M'). We consider one more tree type: 'conceptual' tree (marked as 'C'), which subsumes 'M' or 'F' type trees. A conceptual tree is one that has at least one node with more than two child clusters and more than 80% of the words in each child cluster have similar meaning. An instance is explained in Section 6.1. Since we prefer a tree that can represent meaningful concepts, 'C' type trees are the most desirable. 'T' type trees are degenerate and are undesirable.

Based on these evaluation criteria, we analyze different similarity functions (Sec. 6.1), threshold-finding methods (Sec. 6.2) and window sizes (Sec. 6.3).

6.1 Similarity Function

We compared two similarity functions: *AEMI* versus *AEMI-SP*. We fixed the threshold-finding method to Valley and the window size to 'entire page.' Table 6 illustrates the results. The letter 'U' stands for user, 'Total' means the total number of nodes in the cluster tree, '# of L' means the number of leaf nodes. 'G %' is calculated by dividing the number of 'good' leaves by the '# of L'.

AEMI yielded significantly more meaningful leaf clusters (61% good) than *AEMI–SP* (47%). Both methods generated trees whose shapes are mostly 'medium'. For U8, *AEMI* generated a conceptually related tree. The tree has a node with two child clusters and they contain words from course titles. All the 'C' trees in the other tables are the same as this particular tree. For U2 with *AEMI*-SP, the generated tree is 'fat' and has an ABF value of 10.

AEMI

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	4	4	3	6	4	4	2	6	4	8	8	4	2	59
Good	3	2	2	6	4	3	2	6	2	1	1	2	2	36
Fair	1	2	1			1			2	7	7	2		23
Bad														0
G %	75	50	67	100	100	75	100	100	50	13	13	50	100	**61**
ABF	2.5	2	2	2.7	2	2	2	2.2	2.5	2.4	2.4	2.5	2	
Shape	M	M	M	M	M	M	M	C	M	M	M	M	M	

AEMI-SP

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	10	10	5	10	9	7	7	5	10	13	17	8	4	115
Good	2	6	1	3	3	3	3	3	4	5	6	4	4	47
Fair	8	4	4	7	6	4	2	2	4	5	8	4		58
Bad							2		2	3	3			10
G %	20	60	20	30	33	43	43	60	40	38	35	50	100	**41**
ABF	2.8	10	2.3	3.3	3	3	2.5	3	4	2.7	2.8	3.3	2.5	
Shape	M	F	M	M	M	M	M	M	M	M	M	M	M	

Table 6: *AEMI* versus *AEMI -SP*

6.2 Threshold-finding Method

We compared two threshold-finding methods: Valley versus MaxChildren. We fixed the similarity function to *AEMI* and the window size to entire page. Table 7 illustrates the results. MaxChildren generated more meaningful leaf clusters (61%) than Valley. Tree shapes are similar (medium) in both methods. However, generally, trees generated by MaxChildren are shorter, which indicates that MaxChildren reduces the number of iterations in the DHC algorithm by dividing the cluster in early stage. Hence, MaxChildren is faster than Valley. Most of the trees generated by both methods are 'medium' trees.

MaxChildren

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	4	4	3	6	4	4	2	6	4	8	8	4	2	59
Good	3	2	2	6	4	3	2	6	2	1	1	2	2	36
Fair	1	2	1			1			2	7	7	2		23
Bad														0
G %	75	50	67	100	100	75	100	100	50	13	13	50	100	**61**
ABF	2.5	2	2	2.7	2	2	2	2.2	2.5	2.4	2.4	2.5	2	
Shape	M	M	M	M	M	M	M	C	M	M	M	M	M	

Valley

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	6	6	4	6	5	5	4	3	3	8	11	4	7	72
Good	4	4	1	5	2	3	4	1	1	1	2	3	3	34
Fair	2	1	3	1	2	2		2	2	7	7	1	4	34
Bad		1			1						2			4
G %	67	67	25	83	40	60	100	33	33	13	18	75	43	**47**
ABF	2.7	2	2	2.7	2.3	2.3	2	2	3	2.5	2.4	2.5	2.5	
Shape	M	M	M	M	M	M	M	M	M	M	M	M	M	

Table 7: MaxChildren versus Valley

6.3 Window Size

We compared the performance using different window sizes: 'entire page' versus 100 words (paragraph length). We fixed the similarity function to *AEMI* and the threshold-finding method to Valley.

Window size = entire page

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	4	4	3	6	4	4	2	6	4	8	8	4	2	59
Good	3	2	2	6	4	3	2	6	2	1	1	2	2	36
Fair	1	2	1			1			2	7	7	2		23
Bad														0
G %	75	50	67	100	100	75	100	100	50	13	13	50	100	**61**
ABF	2.5	2	2	2.7	2	2	2	2.2	2.5	2.4	2.4	2.5	2	
Shape	M	M	M	M	M	M	M	C	M	M	M	M	M	

Window size = 100 words

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	Sum
# of L	5	2	12	9	4	4	2	7	8	13	1	6	4	77
Good	5	2	3	5	4	3	2	7	3	2	1	3	4	44
Fair			8	4		1			5	11		3		32
Bad			1											1
G %	100	100	25	56	100	75	100	100	38	15	100	50	100	**57**
ABF	3	2	4.7	3.7	2.5	2.5	2	3	3.3	3.4	1	3.5	4	
Shape	M	M	M	M	M	M	M	M	M	M	T	M	M	

Table 8: Window size of entire page versus 100 words

Table 8 illustrates the results. Window size of the entire page generated slightly more meaningful clusters (61% good) than window size of 100 (57% good). However, window size of 100 yields more tress with 100% good leaf clusters (6) than window size of the entire page (5). Hence, it is clear which window size produces more meaningful clusters. Both methods resulted in trees whose shape we evaluated to be 'medium'. Window size of 100 generated one thin tree for U11. The 'T' tree in Table 8 has only two nodes: the root and one leaf.

7. CONCLUDING REMARKS

To create a more robust context for personalization, we proposed learning a cluster hierarchy that can represent a continuum of general (long-term) to specific (short-term) interests from a set of web pages visited by a user. This approach is non-intrusive and allows web pages to be assigned to multiple clusters. We proposed our divisive hierarchical clustering (DHC) algorithm and evaluated it based on data obtained from 13 users on our web server. We also introduced similarity functions and threshold-finding methods for the clustering algorithm. Our empirical results suggest that the *AEMI* similarity function and the MaxChildren threshold-finding method yielded more meaningful leaf clusters. Using *AEMI* and MaxChildren, DHC generated over 60% interpretable hierarchical clusters.

The window size does not make significant difference; however, we suggest a window size of 100 since usually meaning within one paragraph is more cohesive than within one document. Results from experiments not reported here indicate that stemmed words are more effective than whole words. The minimum cluster size affects the number of leaf clusters and size 4 was easy to use and seemed to produce reasonable results.

Till now, we have considered only single words; phrases may provide more information about topics compared to words. For instance, "apple" has different meanings in "apple tree" and in "apple computer". Phrases can be found using *AEMI* [3]. Since nodes with only one child are undesirable and they could lead to degenerate trees, we can improve HDC to ensure that "single-child" parents do not exist. After the threshold is determined, if only one child exists, the child is not added. We repeatedly apply the threshold-finding method in the "unborn" child until more than one child is produced.

ACKNOWLEDGMENTS

We thank the members of Laboratory for Learning Research (LLR) for their comments.

REFERENCES

1. Bellegarda, J.R. Exploiting both local and global constraints for multi-span statistical language modeling, IEEE Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 2, 677-680, 1998.
2. Billsus, D., and Pazzani, M.J. A Hybrid User Model for News Story Classification, Conf. User Modeling, 1999.
3. Chan, P.K. A non-invasive learning approach to building web user profiles, *KDD-99 Workshop on Web Usage Analysis and User Profiling*, 7-12, 1999.
4. Fisher, D.H. Knowledge Acquisition via Incremental Conceptual Clustering. Machine Learning 2, 139-172, 1987.
5. Frakes, W.B., and Baeza-Yates, R. Information Retrieval: Data Structures and Algorithms, Prentice-Hall, 1992.
6. Han, J. Data Mining Concepts and Techniques, San Francisco : Morgan Kaufmann Publishers, 2001.
7. He, X., and Ding, C.H.Q., (etc). Automatic topic identification using webpage clustering IEEE ICDM, 2001.
8. Pazzani, M., and Billsus, D. Learning and Revising User Profiles: The Identification of Interesting Web Sites, Machine Learning, 27(3), 313-331, 1997.
9. Perkowitz, M., and Etzioni, O. Towards adaptive Web sites: Conceptual framework and case study, Artificial Intelligence 118, 245-275, 2000.
10. Richardson, M., and Domingos, P. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank Advances in Neural Information Processing Systems 14, 2002.
11. Russell, S., and Norvig, P. Artificial Intelligence A Modern Approach. Prentice Hall, 74, 1995.
12. Voorhees, E.M. Implementing Agglomerative Hierarchical Clustering Algorithms for use in document retrieval, Information Processing & Management, 22 (6) 465-476, 1986.
13. Zamir, O., and Etzioni, O. Groper: A Dynamic Clustering Interface to Web Search Results, The Eighth International World Wide Web Conference, Toronto, 1999.
14. Zamir, O., and Etzioni, O. Web document clustering: a feasibility demonstration. In Proc. SIGIR-98, 1998.