Implicit User Interest Profile

Philip K. Chan and Hyoung R. Kim
Computer Science Department
Florida Institute of Technology
Melbourne, FL. 32901

pkc@cs.fit.edu - Philip
goddoes8@hotmail.com - Hyoung

ABSTRACT

User interest profile presents items that the users are interested in. Typically those items can be listed or grouped. Listing is good but it does not 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. This hierarchical user interest profile has obvious advantages: specifying user's specific interests and general interests and representing their relationships.

Current user interest profile structures mostly do not use implicit method, nor use an appropriate clustering algorithm especially for conceptually hierarchical structures. This research studies building a hierarchical user interest profile (HUIP) and the hierarchical divisive algorithm (HDC). Several users visit hundreds of web pages and each page is recorded in each users profile. These web pages are used to calculate HUIP for each user.

Using the web pages from the users, the relations between words are calculated and clusters of words are made hierarchically. We use AEMI function to calculate the similarity of words, which means how much the words are related, and several techniques to calculate threshold to divide clusters to build more accurate and detail profiles.

EXT-2002-065
04/09/2002

1. INTRODUCTION

One way that a user interest profile builder can be intelligent is to recognize interests for a user. The high level goal is to build long term and short term interests of a user; the lower level goal is to build simple list of what they are interested in. Intelligently understanding the interests of a user is critical for many systems that use intelligent user service [pkc].

In order to adaptively recognize interests, a system must know what users are interested in. The most common and obvious solution is for the user to list in an *explicit* way, where users type in to the system what they are interested in (e.g., music, computer, car) or piece of information (e.g., news article). *Explicit* methods are well-understood, fairly precise, and are common techniques in these days.

However:

- Have to have the user mark his/her interests.
- User's interest can keep changing, so asking a user periodically to upgrade is not convenient.
- Sometimes a user does not know what they are interested in [need reference].

Hence, *explicit* way, while common and trusted, may not be as convenient as is often expected, since the system maybe can ask the user every week to mark what they are interested in. As a solution *implicit* way is to build user's interests by a method other than obtaining them 'directly' from the user. The advantages of implicit way are:

- They remove the cost of the user reading and marking.
- The system can automatically update the user's current interests.
- Can be gathered for free.
- Can be combined with other implicit methods for a more accurate profile.
- Can be combined with explicit ratings for an enhanced user profile.

There is another consideration. A profile can be listed in a flat style: listing all of the user's interesting area (e.g., car, computer, monitor). Listing is common and used broadly and is easy to understand.

However:

- It has some restrictions on representing user's long-term and short-term interests [pkc].

Even, books in a library are listed in a hierarchical way. So, the solution is to build a user interest profile in a hierarchical way. It posses 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. This hierarchical user interest profile has obvious advantages: it can specify user's specific interests, general interests and their relations.

We believe that a user interest profile builder, the use of implicit method, and representing them in a hierarchical way has significant benefits yet poses significant challenges that have yet to be investigated.

The main objective of this research is to build HUIP in an implicit way. We concentrate on a set of cluster for a single person. To build a HUIP, we developed an algorithm, called Hierarchical Divisive Clustering (HDC) algorithm. We run this algorithm on several sets of data of different users.

We analyzed the returned clusters and measured the quality of HUIP using statistics. The combination of "Counting cluster" and AEMI similarity function returned a useful hierarchical cluster tree. The window size does not show significant differences; however, we pick window size 100 since it supports our research conceptually better: one page can possess multiple topics. Some leaf clusters that have a similar concept derivate from same parent, so that we can assume that our algorithm supports a conceptually hierarchical cluster tree. Moreover, *implicit* user interest profile builder may be as effective as *explicit* way in terms of accurate coverage while having none of the user-costs from explicitly asking the interests.

The contributions of this work are:

- Algorithm that builds a conceptually hierarchical cluster tree.
- New divisive clustering algorithm.
- A set of user interest profile.

The rest of this paper is as follows: Section 2 describes related work in clustering algorithm and building user interest profile; Section 3 describes a general categorization of user interest profile; Section 4 details our approach towards building implicit user interest profile; Section 5 describes our experiments; Section 6 analyzes the results from the experiments; Section 7 presents our conclusion; and Section 8 mentions some possible future work.

2. RELATED RESEARCH

We have divided related work in user interest profile into two categories: work that describes new hierarchical divisive clustering algorithm, and work that builds user interest profile related to web personalization.

2.1 Existing Clustering Algorithms

HAC is an agglomerative hierarchic clustering algorithm; such as single link clustering algorithms, complete link method, and group average link method are the members of it. The single link clustering algorithms has a running time of $O(N^2)$ and a space requirement of O(N). In the single link method the similarity between two clusters is the maximum of the similarities between all pairs of words such that one word of the pair is in one cluster and the other word is in the other cluster.

The definition of the complete link method is just the opposite of single link clustering algorithms. The similarity between two clusters is the minimum of the similarities between all pairs of words such that one word of the pair is in one cluster and the other word is in the other cluster. The similarity between two clusters in the group average link method is the mean of the similarities between all pairs of words such that one word of the pair is in one cluster and the other word is in the other cluster [1].

The K-mean algorithm selects an initial partition with *K* clusters. An initial partition can be formed by first specifying a set of *K* seed points which can be the first *K* patterns or *K* patterns chosen randomly from the pattern matrix. Different initial partitions can lead to different final clusters because algorithms based on square-error can converge to local minimum. This is especially true if the clusters are not separated well. As a second step it generates a new partition by assigning each pattern to its closest cluster center and computes new cluster centers as the centroids of clusters. A set of *K* patterns that are well separated form each other can be obtained by taking the centriod of the data as the first seed point and selection successive seed points which are at least a certain distance away from the seed points already chosen. As a third step, it repeats second step until an optimum value of the criterion function is found. Partitions are updated by reassigning patterns to clusters in an attempt to reduce the square-error. The Euclidean metric is the most common metric for computing the distance between pattern and a cluster centers. As a final step, it adjusts the number of clusters by merging and splitting existing clusters or by removing small, or outlier, clusters [2].

K-mean is fast, popular, and statistical clustering. The previous research [3] tested both algorithms. In the case of *K*-means, they set *K* much larger than their target, ten clusters. They tuned *K* to produce the desired cluster number. In the case of HAC, they also set a parameter *K* much larger than their target cluster number. In the previous research K-means shows better result, but HAC is normally famous for its high quality. The complete link hierarchy is useful for searching cliques. The group average link hierarchy can be a hierarchical cluster tree by itself while the other two single and complete links are the set of clusters obtained by varying threshold over all distinct weights in the similarity matrix. It needs a function to merge centroids, and that can affect the accuracy. Since we are more interested in the cluster that are not strict and cluster the elements well than strict clique, we disregard complete link. K-mean uses certain *K* parameter to cluster and tune the *K* for best results, starting from big number and gradually reducing the number [3].

2.2 Hierarchical User Profile

* AUTOCLASS
* COBWEB

3. USER INTEREST HIEREARCHY

Our research falls into a category of using implicit method and building hierarchical user interest profile. The most basic is to consider them on an Implicit/Explicit dimension, as described in figure 1. This dimension is based on how much information about their interest the user provides specifically. Another beneficial view is to consider 'what the structure of the profile is'. It can be flat or hierarchical or any other. This is related to the shape of a cluster.

Explicit User specifies items

Implicit No user intervention

Figure 1: Explicit/Implicit Dimension of Interest

The input of our method is a set of web pages and output is conceptual hierarchical clusters for a person. We use only the words in a web page not link and image information. The web pages are stemmed and removed by stop lists. Output is a set of clusters with hierarchical shape: the top cluster keeps all of the words and it's child clusters inherit some words out of the words in the parent cluster. For instance in table 1 we present some artificial data set. Numbers in the left represent individual web pages stemmed and removed by stop list. These web pages can be represented by a conceptually hierarchical cluster tree shown in figure 2. Each siblings has conceptual relations, for example 'perceptron' and 'backpropogation' can be categorized to 'Neural network algorithm', but 'ID3' and 'C4.5' does not; however, they all belong to a 'machine learning' category.

Our idea is based on that some related words could be connected by some key words such as 'data mining' and 'machine learning' would occur more frequently with 'ANN', 'perceptron', 'backpropergation', 'decision tree', 'ID3', 'C4.5', and 'hypothesis space'; however, the *italic* words may not occur along with *caps* words in Figure 2. Therefore, the words, 'data mining' and 'machine learning', do the role of a key words that connects *italic* and *caps* words.

page	Contents
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 BFS
9	Ai searching algorithm forward checking constraint reasoning
10	Ai searching algorithm forward checking constraint reasoning

Table 1: Artificial data set

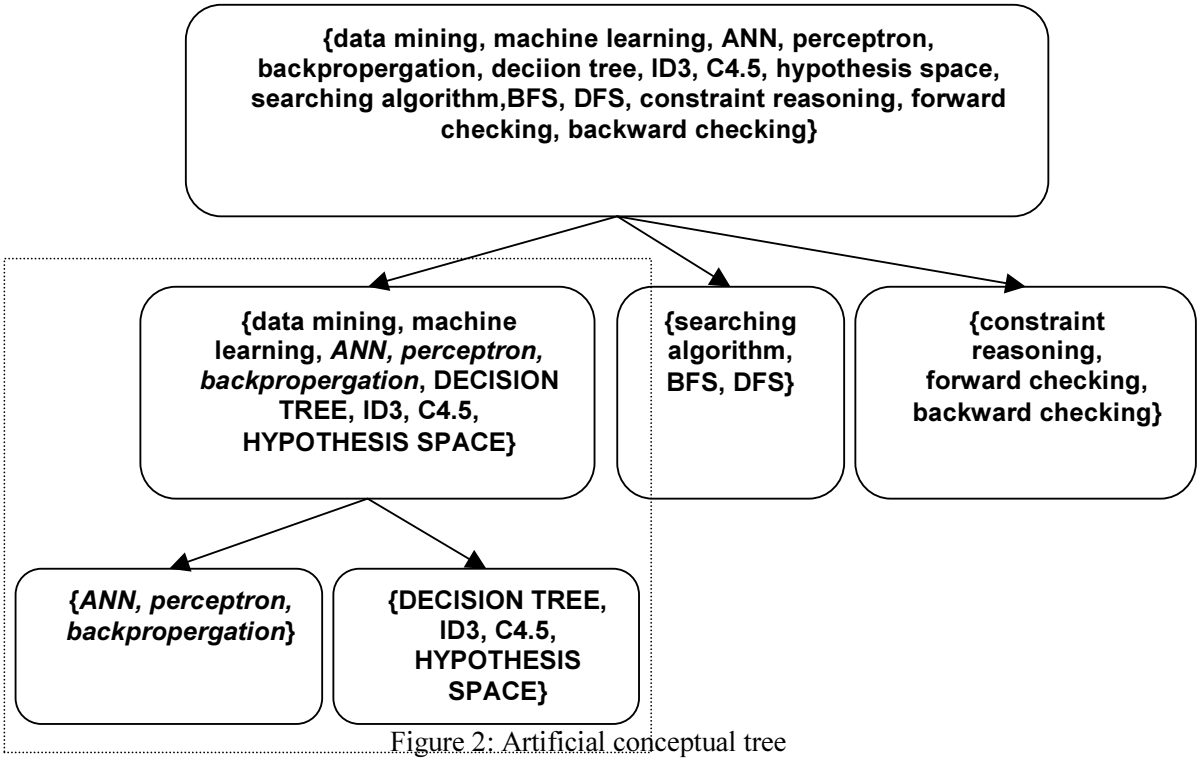

Figure 2: Artificial conceptual tree

4. APPROACH

Our approach is to experimentally measure and analyze our algorithm, and parameters presented below, in order to ascertain their effectiveness. We used following methodology:

- Implement an algorithm to cluster the data.
- Run the program with several user's data sets.
- Analyze the clusters returned.

This section details the algorithm we implemented, called HDC, to build HUIP from the recorded data sets as they browsed the web. The HDC algorithm provides some variables: 'similarity function' that you can change for other ones, 'finding threshold' where you can apply various methods, and window size.

4.1 Algorithm

The complexity, efficiency, and accuracy depend on how algorithm works heavily. This algorithm generates hierarchical cluster. The algorithm operates by iteratively adding clusters to the Cluster List. Examples, web pages collected (we treat only words not grammatical tokens like HTML or XML), are first removed by stop lists and stemmed. Vocabulary is a cluster with distinct set of words. On each iteration similarity value between words are calculated and stored in an AemiTable variable, new threshold of the cluster is calculated using various threshold-finding methods. Then all the connected words, which mean the similarity value in between is bigger than the threshold value, are grouped and build another Vocabulary. A new Vocabulary is divided into child vocabulary recursively using depth first algorithm. Some vocabulary smaller than the minimum cluster size are dismissed. A Vocabulary keeps page information, which means the pages numbers that a word occur are recorded.

CalculateAemiTable function gets SIMILARITYFUNCTION, Vacabulary, and Window size as parameters and return AemiTable, where the Window size affects the conjunction value of words by measuring the distance between words. CalculateThreshold function gets FINDTHRESHOLD and AemiTable as parameters. We apply various threshold-finding methods to calculate better threshold.

UIH (Examples, SIMILARITYFUNCTION, FINDTHRESHOLD, WindowSize)

Examples: A set of web pages, we treat only words not grammar.
SIMILARITYFUNCTION: A similarity function
FINDTHRESHOLD: A method to cut a threshold
WindowSize: Window size which is used to measure the distance between two words and if they are too far from each other they are not considered as being in a same page

1. Remove stop lists from examples
2. Stem Examples
3. Vocabulary ← The set of all distinct words occurring in any text documents from Examples.
4. ClusterList ← **HDC** (Vocabulary with page info)
5. Return ClusterList

HDC (Vocabulary)

1. Stopping condition:
   If the number of Vocabulary is smaller than or equal to minimum cluster size.
   If it repeats without reducing the number of words in a Vocabulary.
2. AemiTable ← **CalculateAemiTable** (SIMILARITYFUNCTION, Vocabulary, WindowSize)
3. Threshold ← **CalculateThreshold** (FINDTHRESHOLD, AemiTable)
4. While (Vocabulary ← Get a connected words by using Depth First Search)
   - ClusterList ← ClusterList + Vocabulary
   - **HDC** (Vocabulary)
5. Return ClusterList

CalculateAemiTable (SIMILARITYFUNCTION, Vocabulary, WindowSize)

1. for each word *a* in a Vocabulary
   for each word *b* in a Vocabulary
   a∧b ← Calculate a∧b; however, if a and b are farther than window size
   they are not counted.
   AemiTable ← SIMILARITYFUNCTION (a, b, a∧b, ~a∧b, a∧~b, ~a∧~b)
2. Return AemiTable

CalculateThreshold (FINDTHRESHOLD, AemiTable)

Return ← Apply FINDTHRESHOLD method to AemiTable and calculate threshold

---

Figure 3: HDC algorithm

4.2 Similarity Function

Similarity function explains how two words are close or far. If a proper similarity function can be found it will help the accuracy of this system. We are first proposing four similarity functions and comparing the validity theoretically and empirically. Those are AEMI (Augmented Expected Mutual Information), AEMI × Specification, Jaccard function, and MAX function. And we also present specification function to weight each similarity value.

1) AEMI Function

AEMI is enhanced version of MI (Mutual Information) and EMI (Expected Mutual Information), which appropriately incorporates the counter-evidence. The previous research uses AEMI to search multi-word features by co-occurrence of the words, that is, if two words appear together frequently, they are assumed as a phrase in that research [4]. In our research we are finding out the words that occur in a same document frequently and assume they are related each other. So we define A as the ratio of the documents that contains word a from the whole documents, and B as the probability of second word occurring at the same document. Therefore the higher value of AEMI indicates a is likely occur with b in the same document and the lower one is less likely to do when the other is absent. The final formula is

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

2) AEMI × Specification Function

We combine AEMI and Specification function that has the information of common or specific of the word in the cluster. We propose specification function (SP) that weights each similarity value. The behind idea is that we want to assign higher value to more specific concept words, so that we prefer those words that are not common. We use devised sigmoid function. The x value is the max value out of the two probability values of two words. We draw the graph where x axis has the value from 0 to 1, and the return value also has the value between 0 and 1.

$$y = 1/(1 + \exp(0.6 * x)$$

3) Jaccard Function

This is a simple function so it is easy to understand and make meaningful cluster but they was not hierarchical so we stop using this function. When we calculate the AEMI table out of the sample data, the related value of ‘ai’ is supposed to be very small, since the words are very predominant; however, those values are bigger than average, which makes us hard to make child clusters, that means it is not proper for making hierarchical clusters.

$$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$$

4) MAX Function

MAX function takes bigger value between P(A|B) and P(B|A). The behind idea is that if we assign a same similarity value with connected words and connecting words, they would go together, since they are going to be chose. However in real test this does not work well.

$$MAX (P(a,b)/P(a)), D(P(a,b)/P(b)))$$

4.3 Finding Threshold

The threshold value divides strong edges from weak edges in an AEMI table. If the threshold value were very low it would connect all words, on the other hand if the threshold value were very high it would not connect any words. Given the AEMI table we tried to find the reasonable threshold values that can cluster the data set in best way. We propose five different methods: positive biggest gap, edge cut, average + standard deviation, deep valley cut, and counting clusters.

- Positive biggest gap

We tried to find out best threshold leading to best cluster results. The first idea was using gap information; it seemed that the biggest gap divide the AEMI value into the valid ones from others. We disregarded the negative value and found biggest gap within the positive AEMI values. When it meets the biggest gap too close from the positive biggest value it lost too much information and returned very little cluster tree.

- 30% edge cut

We wanted to solve the high threshold problem and at first used bruit force method, which is just to find the threshold that includes around 30% of the high threshold value. It gave more information than above method but has low threshold problem which make talk thin tree.

- Average + standard deviation

To increase the threshold value we calculated average + standard deviation of valid AEMI values. By examining the test data and AEMI table we found out another truth that we do not need to cluster the words that exists only one page. It returned reasonable results from both test data sets, but it was unstable.

- Deep valley cut

To divide strong edges from weak edges we examine the group of edges whether there are gaps that divide them. Our technique is that first we make histogram with the number of edges and the under bound and upper bound. We find deepest valley; in addition to that we also find biggest gap within the deepest point. We assume the edges could be grouped strong relations and weak relations by this method.

- Counting clusters

We count the cluster numbers with various threshold values and pick the most divisive threshold. Some big clusters still disappear suddenly with previous method. The reason is that the threshold is too high, which means there is information loss. We cannot say that these thresholds are best; however, since we consider over 60% of the edges these are high enough. Furthermore,

it guarantees the division once they can be divided. The main idea of preferring large number of child clusters is that it seems to explain a human concept in more detail.

4.3 Window Size and Minimum Size of a Cluster

One web page or one document can include several contents. Therefore regarding them as one concept could cause confusion. So we try to divide a page paragraph by paragraph or sentence by sentence. So we vary the window size: without limitation, 100, and 30. The number 30 is the number of words that a person can read in 15 second. The 15 seconds is the time a person can concentrate on reading in a web page [need reference]. In our experiment the window size does not make significant differences.

The minimum size of a cluster affects the number of clusters and threshold. Too much clusters are hard to read and can cause computational delay when they are used in other application. So we picked number 4 that can represent a concept and big enough to make less number of clusters.

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 accessed at least 50 times in the first two months and also in the second 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. We picked 13 different users and collected the web pages they visited. The machine where we run this system is Sun Ultra 60. The author valued each clusters by counting number of reasonable and related words within a cluster, in other words perceptibility and relationship.

6. ANALYSIS

The implicit user profile methods we analyze in this section are:

1. Meaningful leaf clusters and relationship with parent clusters (Section 6.1).
2. Shape of the hierarchy (Section 6.2).
3. Window size (Section 6.3).

Initially, we analyzed 'Deep valley cut' versus 'Counting cluster' methods. We focus on the 'good' results since the purpose of our research is to find good clusters. We marked a cluster 'good' if there exist any relations between words or they represent some meaning. We marked a cluster 'bad' if it is very hard to judge a cluster because its size is very big or of some other reasons. We divided 'Good' by 'Total' and represent '% of good'.

6.1 Finding Threshold

We compared two threshold-finding techniques: 'Deep valley cut' versus 'Counting cluster'. The comparison of the % of meaningful leaf clusters turns out that Counting cluster returns better results: total average is 61% of 'good' with Counting cluster and 47% of 'good' with 'Deep valley cut' shown in figure 4. We also checked the shape of the tree. The both shapes are similar and hard to tell the differences, but generally the tree of 'Counting cluster' is shorter, which means 'Counting cluster' reduces the number of iterations by dividing the cluster in early stage. As for the speed of the experiment the 'Counting cluster' is faster than 'Deep valley cut' since it reduces the iteration number by counting the number of cluster.

Counting cluster

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	75%	50%	67%	100 %	100 %	75%	100 %	100 %	50%	13%	13%	50%	100 %	61%

Deep valley cut

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	67%	67%	25%	83%	40%	60%	100 %	33%	33%	13%	18%	75%	43%	47%

Figure 4: Counting cluster versus Deep valley cut

6.2 Similarity Function

We compared two similarity functions: 'AEMI' versus 'AEMI × Specification function'. Similarity function with 'AEMI' returns better results than 'AEMI × Specification function': 61% of good with 'AEMI' and 47% of good with 'AEMI × Specification function' shown in figure 5. So that we see that the specification value as a weight confuses the algorithm.

AEMI

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	75%	50%	67%	100 %	100 %	75%	100 %	100 %	50%	13%	13%	50%	100 %	61%

AEMI × Specification Function

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	20%	60%	20%	30%	33%	43%	43%	60%	40%	38%	35%	50%	100 %	41%

Figure 5: AEMI versus AEMI × Specification Function

6.3 Window Size

We compared the results of different window size: 'without limitation' versus 100. When we summed the number of good clusters and calculate the % of 'good' the result from 'without limitation' is better, without limitation is 61% and window size 100 is 57%. However, generally window size 100 makes more better clusters, without limitation makes 5 100% clusters and window size 100 makes 6 100% clusters. So, it is hard to tell which one is better than the other.

Window size without limitation

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	75%	50%	67%	100 %	100 %	75%	100 %	100 %	50%	13%	13%	50%	100 %	61%

Window size 100

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	Total
Total	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
% of good	100 %	100 %	25%	56%	100 %	75%	100 %	100 %	38%	15%		50%	100 %	57%

6.3 Summary

In this work, we developed a new algorithm, HDC, and tested some other parameters in order to build implicit user interest profile. This algorithm can be combined with other algorithms, such as estimating page interest or some speech recognition tools that detects what the speaker

said. Using this method of client-side implicit user interest profile builder may help building a user's interest profile; however, the server-side one may also help to build interest profiles for each visitor.

We validated the efficiency of the HDC algorithm and find some efficient parameters: 'Counting cluster' method for threshold- finding method, 'AEMI' for similarity function. The combination of 'Counting cluster' method and 'AEMI' function yields over 60% accurate hierarchical clusters, which we can say practical. The window size does not make significant difference; however, we pick window size 100, since it is more reasonable conceptually: one paragraph has one meaning, and one paragraph has around 100 meaningful words without stop lists.

We also measured some other parameters. The use of stemmed words returned better results then the one of un-stemmed words. The minimum cluster size affects the number of child clusters, and size 4 was easy to use.

We also find this algorithm supports conceptual user interest hierarchy. Some clusters have conceptually similar meaning, such as a cluster has class names and the other sibling has the other class names. Since they derivate from same parents we assume they are connected conceptually. Even though this kind of hierarchy only occurs a few times, we satisfied with identifying whether this algorithm supports this conceptuality or not. To statistically prove this we may need more laboratorial research.

7. CONCLUSIONS

One way a user interest profile can be automated is to analyses the web pages that a user visited. Explicit methods, such as asking users to mark their interests, bothers the users and the system should collect the information periodically. Sometimes the information collected may be not correct as much as we are expecting by the negligent marking of the users. Implicit method, while requiring more sophisticated algorithm, promises to provide user interest profile without the "cost" to the users reading, thinking, and marking periodically.

In this research we have experimentally evaluated the effectiveness of threshold- finding methods, similarity functions, and window size in building accurate user interest profiles. Based on the data collected from 13 users, we find that 'Counting clusters' threshold- finding method, 'AEMI' similarity function are good methods for clustering user interest hierarchy and the algorithm HDC with depth first method is also proper for this work.

The techniques used in this research provide a means of building implicit user interest profile. This could be combined with other techniques such as a technique to estimate page interest, but also be combined voice recognition techniques to present what subject a user is interested in. The results presented promise to strengthen the accuracy by today's automatic method building user interest profile and introduce new user interest profile technique.

8. FUTURE WORK

In this work, we have considered only the word alone; phrases may provide more accurate information than words. Such as "apple" has various meaning: "apple tree" and "apple computer", so phrases may explain some more detail meanings. The set of stop list also affects the accuracy of the results. So specializing the stop list can be a future work.

Future work also suggests better similarity function that accurately explains relationship between words. And other better threshold-finding method or the other parameters will help building HUIP. While our intent here was to building HUIP it will be combined to a technique that estimates page interest.

9. ACKNOWLEDGMENTS

The algorithm and results of the experiment can be downloaded from http://.

10. REFERENCES

[1] Ellen M. Voorhees, "Implementing Agglomerative Hierarchic Clustering Algorithms for use in document retrieval", Information Processing & Management, Vol. 22, No. 6, pp. 465-476, 1986.

[2] "K-mean algorithm", http://www.cs.uregina.ca/~linhui/K_mean_algorithm.htm (* will be fixed in a right format later)

[3] Mike Perkowitz, Oren Etzioni, "Towards adaptive Web sites: Conceptual framework and case study", Artificial Intelligence 118, pp. 245-275, 2000.

[4] Philip K. Chan, "A non-invasive learning approach to building web user profiles", KDD-99 Workshop on Web Usage Analysis and User Profiling.