Constructing Personal Digital Library by Multi-Search and Customized Category

[1]Ching-Chi Hsu, [2]Fan-Chen Tseng , [1]Ji-Ming Chen, [1]Chun-Hung Chang
[1]Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106
[2]Department of Electronic Engineering
National I-Lan Institute of Technology
I-Lan, Taiwan 260

Abstract

The current search tools for retrieving information on WWW are not suitable for building customized information repository because these search tools are designed for general users with the result of only an unstructured collection of documents. In this paper, we present a personal digital library capable of efficiently retrieving information on the World Wide Web. We adopt several new strategies to overcome the shortcomings of current tools. The first strategy, Classification, merges and organizes the retrieved documents to put them in a structural, hierarchical frame. The second strategy, User Profile, saves time and bandwidth for the access of the documents and permits the users to build their own customized category structure. The third strategy, Multi-Search, capitalizes on the power of multiple search engines to broaden the domains of information sources and alleviate the overloading of a single search engine. Furthermore, we derive in detail the techniques for speeding up the iterative process of clustering.

Keywords: *digital library, multi-search, classification, clustering, user profile*

1. Introduction

World Wide Web (WWW) has created a new era of information. More and more web sites are set up to provide information about academics, business, education, entertainment, and government. Similarly, more and more people retrieve information from WWW with inexpensive personal computers. To both business organizations and individuals, WWW represents a prospective frontier of opportunities.

Information Retrieval (IR) on WWW, nevertheless, is suffering from three major shortcomings [1,6,9]. The first problem is single-engine search. The massive contents of WWW and the intensive daily information need of tens of thousands of people make it difficult for a single search engine to efficiently retrieve information on WWW for every single user. The second inconvenience for personal users is the fact that the retrieved information is not well organized. The user has to take pains searching through the heterogeneous documents to find relevant ones. The third inefficiency is due to the lack of customization. The search engine is not specialized for personal use; consequently, the history and traits of personal information need are not utilized for information retrieval. All these three problems must be solved to construct an efficient Personal Digital Library.

For each of the above problems, this paper proposes a solution. To conquer the first problem (single search engine), a multi-search strategy [8] is adopted to send the query to several search engines. This way, the search load is spread among these engines and at least two benefits ensue. First, since each search engine has its characteristic domain of information source, the multi-search approach results in broader domains of information. Second, since the query is sent simultaneously to several engines, even when some engines is busy, some other engines may be returning the documents. As a result, the

0-7803-5214-9/98/$10.00 © 1998 IEEE.

documents come out at a stable rate.

To solve the second problem (heterogeneous documents), a classification process [2] is adopted. The retrieved documents come from different search engines which have their own categorization philosophies; consequently, the original retrieved documents are presented in a heterogeneous way. In our system, the classification scheme is based on a database of massive documents to establish an unbiased category tree. Therefore, after classification, the user can view the documents in well-organized structures.

As for the third issue (lack of customization), a user profile is set up. The user profile keeps track of the relevant information of query results and lets the users construct their customized categorization hierarchy.

2. System Architecture and Functions

2.1 System Architecture and Principles of Operation

The architecture is shown in Figure 2.1. **IR-Client** is a Java Applet that works in Java virtual machine like Netscape Navigator. It is the user interface which communicates with **IR-Server** through socket module. **IR-Server** is a Java Application. It contains five major parts to be discussed later. **Category Database** contains a category tree and a term weight table to be used in classification process. It is maintained by a MiniSQL server. **Profile Database** keeps track of the relevant information of query results. It is implemented with Microsoft Access on a personal computer. **URL Data Structure** is used to temporarily hold the associated information of retrieved documents to be referenced by **Classification Module** and **User Profile Manager.**

The five major components of **IR-Server** consist of **Central Control Program, Multi-Search Module, Classification Module, Categorization Manager,** and **User Profile Manager.**

The **Central Control Program** is a message dispatcher and coordinator. It gets messages from **IR-Client** and dispatches these messages to other components. Besides, it controls the operation flow of other components.

The **Multi-Search Module,** upon receiving a query, directs the query to several specific search engines (such as AltaVista, Excite, Infoseek, Lycos, Magellan, and WebCrawler, etc.) to search and fetch the relevant documents, and stores the associated data of the retrieved documents in **URL Data Structure.** At the same time, the **Categorization Manager** selects from the category tree of **Category Database** a number of categories that

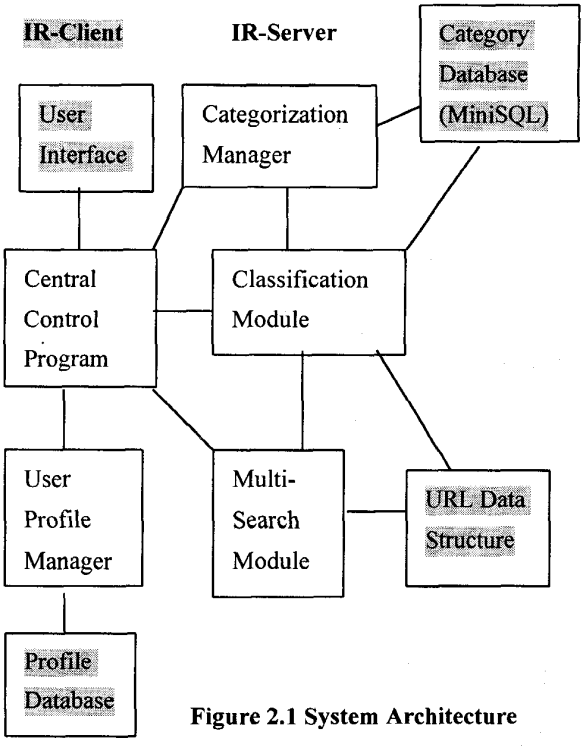

Figure 2.1 System Architecture

are relevant to that query. These selected categories are called *Candidate Categories*.

After that,the **Classification Module** classifies the retrieved documents into the *Candidate Categories*. The user can then view the documents in well-organized structures. Moreover, the user can trace down the subcategories of a category (this function is called **Focus**) to make the Classification Module reclassify the documents into these subcategories, which are provided by the **Categorization Manager**. All the whole process is called *New Query Mode.*

However, the usual operation may get into trouble in two cases; both are with the **Categorization Manager.** The first case occurs if, for some query, there exists no corresponding *Candidate Category.* Therefore, the retrieved documents will be unclassified. The second case occurs when the user attempts to trace down a leaf node of the category tree and the **Categorization Manager** is unable to provide subcategories. For these situations, our system establishes a **Clustering Module** to generate clusters with the retrieved documents themselves. The **Clustering Module** is included in the **Classification Module.** Whenever the classification runs into the above two cases, the **Clustering Module** is invoked.

While retrieving documents from WWW in *New Query Mode*, the user can ask the **User Profile Manager** to save the documents and the associated information

(URLs, titles, category, keywords, etc.) in local **Profile Database** to construct a customized category. Later, the user can search and retrieve documents in the **Profile Database** with **User Profile Manager** in *User Profile Mode.*

2.2 Functions

There are four functions in this system: **Search, Profile, Freeze,** and **Focus**. These functions can be activated by clicking the command buttons of the User Interface shown in Figure 2.2.

2.2.1 Search. Search can be done in *New Query Mode,* as shown in Figure 2.2, or in *User Profile Mode,* as shown in Figure 2.3. In *New Query mode*, the IR-Server directs the user's query to several specific engines. The retrieved documents are classified into hierarchical candidate categories. On the other hand, in *User Profile mode*, users can search documents in the Profile database.

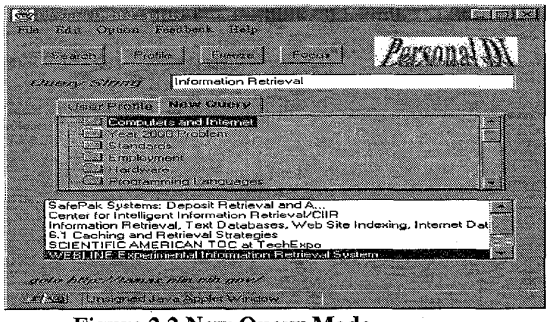

Figure 2.2 New Query Mode

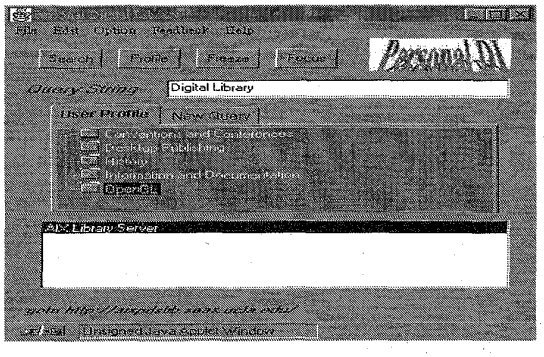

Figure 2.3 User Profile Mode

2.2.2 Profile. The User Profile Manager keeps a customized category structure in local Profile Database, and the Profile function allows users to browse all the stored URL information in this category structure.

2.2.3 Freeze. In addition to providing the candidate categories for multi-search in *New Query Mode*, the **Categorization Manager** gives the user the freedom to select specific members from the candidate categories to reclassify the documents into a narrower range of categories. This function is called **Freeze.**

2.2.4 Focus. In *New Query Mode*, the user can trace down the subcategories of a candidate category with this function. The subdirectories are supplied by the **Categorization Manager**.

3. Design Details

3.1 Categorization Manager

There are three functions performed by the **Categorization Manager** that enable the users to search documents in *New Query Mode.* The first function, **candidate category selection,** selects categories relevant to a new query for classification of the retrieved documents. The second function, **subcategory,** traverses down the category tree to fetch the subcategories of a selected candidate category for the **Focus** function described in section 2.2.4. The third function, **freeze,** fixes a number of candidate categories for classification, as mentioned in section 2.2.3.

3.2 Multi-Search Module

The Multi-Search Module shown in Figure 3.1 is composed of three function units: *Search*, *Fetch*, and *HTML Parser*.

The Multi-Search Module is a multi-thread module (each thread is called a robot) sending the query to several search engines simultaneously. Each engine is assigned a *Search* thread, which uses BFS (Breadth First Search) algorithm to fetch the resulting pages from the engine server. The resulting pages contain the URLs and associated information of the relevant documents, such as titles, dates, and descriptions. After saving this information in **URL Data Structure,** the *Search* thread signals a corresponding *Fetch* thread to fetch the full documents all over the WWW and store these documents in **URL Data Structure**.

When all the relevant documents are fetched, *HTML Parser* makes a token analysis of these documents to produce the < keyword, term frequency > pairs for each document and puts the results in **URL Data Structure**.

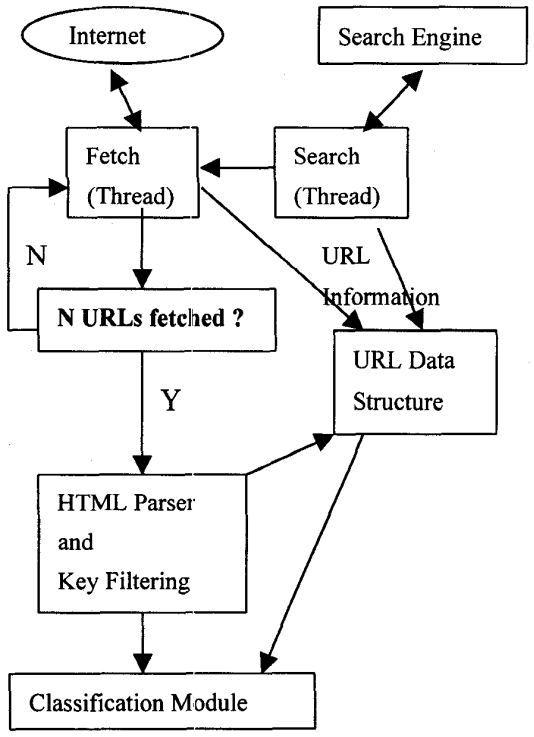

Figure 3.1 Multi-Search Module

3.3 Classification Module

This module needs three resources: (1) Candidate Categories from **Categorization Manager**, (2) URL information composed of <keyword, term frequency> pairs from **Multi-Search Module,** and (3) the hash-table of keyword weight from **Category Database,** which is an unbiased weight dictionary generated from a bulk of documents fetched from WWW.

To classify the documents into candidate categories, a measure function has to be defined to indicate the similarity between a document and a category. In our system, the measure function for the similarity between document k and category j is defined as:

$$M(Doc_k, Categ_j) = \sum_{\substack{key_i \subset Category_j \cap \\ key_i \subset Document_k}} Tf_k(key_i) * Wt_j(key_i)$$

, where $Tf_k(key_i)$ is the term frequency of term i in document k and $Wt_j(key_i)$ is the weight of term i in category j.

With this definition, the algorithm for classification can be expressed as follows:

For each doc:

1. For each candidate category, compute the measure function
2. Find the category which maximizes the value of the measure function
3. Assign the document to this category

3.4 Clustering Module [4,7]

As has been discussed in section 2.1, there are two occasions to use the **Clustering Module.** It can generate clusters with the retrieved documents themselves in these cases. There are five steps for clustering as explained in the following:

Step1: Constructing term frequency table

The pairs of term and term frequency of the documents in **URL data structure** are used to construct the term frequency table such that for each term, the document containing that term and its term frequency in that document can be found.

Step 2: Constructing term weight table

With the results of the first step, the TF (term frequency) and IDF (inverse document frequency) metrics, and a normalization process, a term weight table can be set up so that for each term, its term weights in the documents containing that term can be found.

Step 3:Constructing initial cluster similarity matrix

Initially, each document is regarded as a singleton cluster, so the initial similarity matrix represents the similarity between each pair of documents (the similarity between a document and itself is, of course, one.). Still, the clustering module will interpret the matrix elements as the measures for the similarities between these (singleton) clusters. Therefore, the matrix is better termed "cluster similarity matrix"

Step 4: Merging the two disjoint clusters with highest similarity

The two disjoint clusters with highest similarity are merged to form a new cluster.

Step 5: Updating the similarity matrix

When two disjoint clusters are merged to form a new cluster, the similarities between this new cluster and the other clusters must be recalculated to update the similarity matrix.

Step 4 and step 5 are performed by iteration until the number of clusters are reduced to a specified value or the similarities between the remaining clusters fall below a threshold.

3.5 Profile Database and User Profile Manager

The schema of the Profile Database is shown in Figure 3.2. There are four entity tables: QueryTable,

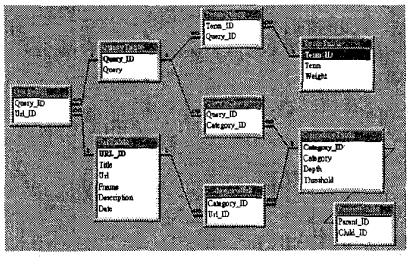

Figure 3.2 Schema of Profile Database

TermTable, UrlTable, and CategoryTable. There are five relation tables: TermQuery, QueryUrl, QueryCategory, CategoryUrl, ParentChild. Note that CategoryTable and ParentChild recursively construct a category tree.

The **User Profile Manager** provides three functions : **url_feedback, init_treeView , and query_profile.**

(1) **url_feedback**

During the search process in *New Query Mode,* the user can save the document information (URLs, titles, etc.) in the local Profile Database. The **url_feedback** function will fill in the attribute entries of the corresponding tables in the **Profile Database.**

(2) **init_treeView**

With the help of this function, the user can browse all URL's in Profile Database in a category organization. The **init_treeView** shows the category tree structure by looking up the CategoryTable, ParentChild, and the CategoryUrL tables of the **Profile Database.**

(3) **query_profile**

This function allows the user to search documents of the **Profile Database** in *User Profile Mode.* The **uery_profile** function will look up the tables of **Profile Database** to fetch the relevant documents.

4. Vector Model for Clustering

The classification and clustering of this system are based on **vector model with cosine measure [3,5]**. This section gives the mathematical background of these techniques.

Before going forward, a number of notations must be define:

1. $\boldsymbol{C}$ denotes a collection of documents; that is, the universe of discourse.

2. Assume $|\boldsymbol{C}| = N$. (There are totally N documents in $\boldsymbol{C}$.)

3. V denotes the set of unique keywords in $\boldsymbol{C}$.

4. Lower case Greek letters ($\alpha, \beta \ldots$) denote individual documents in $\boldsymbol{C}$.

5. Upper case Greek letters (Γ, Δ, Λ) denote document groups; that is, the subsets of $\boldsymbol{C}$.

4.1 Vector Representation of Documents

4.1.1 term weight. For convenience of discussion, let the documents in C be ordered and denoted as α_1, α_2, α_3, α_4 .... α_i, ... α_N, and the terms in V is also numbered from 1 to $|V|$. The term weight of term k in document α_i,denoted by $W_{k,i}$, is defined as:

$W_{k,i} = \log(1+TF_{k,i})*\log(N/DF_k)$, where

$TF_{k,i}$ is the term frequency of term k in document α_i and DF_k is the document frequency of term k in $\boldsymbol{C}$

N/DF_k is the *inverse document frequency (IDF)* of term k in $\boldsymbol{C}$, which is defined to reflect the fact that if a term appears in more documents, that term will then contributes less in distinguishing a document from others.

4.1.2 normalized vector representation of documents [3,5]. With the term weight defined, each document in C can be represented by a $|V|$ – dimensional vector whose components are the term weights of the terms in the document. Specifically, document α_i can be denoted as $\alpha_i = (W_{1,i}, W_{2,i}, \ldots W_{k,i}, \ldots\ldots, W_{|V|,i})$; therefore, the document collection C can be regarded as a *Document Space* consisting of the document vectors.

However, this representation is biased toward longer documents where the number of distinct terms is possibly larger and the term frequencies are possibly higher, which makes longer documents larger vectors. To fix this problem, all vectors in the *Document Space* must be normalized to unit vectors. This, in effect, projects the entire document space onto a $|V|$ – dimensional sphere of unit radius around the origin of the vector space [5]. The normalization factor for document α_i is

$$NF_i = \sqrt{W_{1,i}^2 + W_{2,i}^2 + \ldots + W_{|V|,i}^2} = \sqrt{\sum_{k=1}^{|V|} W_{k,i}^2}$$

The normalized vector representation for document α_i (called its *profile,* denoted by *profile*(α_i)) is a unit vector ; i.e.,

profile(α_i) =($W_{1,i}, W_{2,i}, \ldots W_{k,i}, \ldots W_{|V|,i}$)/$NF_i$

In general , for any document α in $\boldsymbol{C}$, the profile of α is $P(\alpha) = (P(\alpha)_1, P(\alpha)_2, \ldots P(\alpha)_k, \ldots P(\alpha)_{|V|})$, with $P(\alpha)_k$ denoting the normalized term weight of term k in α. Since $P(\alpha)$ is a unit vector, we have $\|P(\alpha)\|=1$.

The notion of profile can be extended to document groups. Let Γ be a document group, the *unnormalized sum profile* for Γ is the sum of profiles of the documents belonging to Γ. In notation, it is

$$P(\Gamma) = \sum_{\alpha \in \Gamma} p(\alpha)$$

4.2 Cosine Similarity Measure

The similarity between two documents α and β *is defined to* be the **cosine** between their *profiles*, which is equal to their inner product since the profiles are unit vectors. In mathematical form, the similarity measure between two documents α and β is

$$S(\alpha, \beta) = < P(\alpha), P(\beta) >$$

$$= \sum_{k=1}^{|v|} p(\alpha)_k * p(\beta)_k$$

It is apparent that $S(\alpha, \alpha) = < P(\alpha), P(\alpha) > = \| P(\alpha) \| = 1$; that is, the similarity between a document and itself is 1.

The *average pair-wise similarity* for a document group Γ (now called a cluster) can be defined to be

$$S(\Gamma) = \frac{1}{|\Gamma|(|\Gamma|-1)} \sum_{\substack{\alpha \in \Gamma \\ \beta \in \Gamma}} \sum_{\alpha \neq \beta} S(\alpha, \beta)$$

$$= \frac{1}{|\Gamma|(|\Gamma|-1)} \sum_{\substack{\alpha \in \Gamma \\ \beta \in \Gamma}} \sum_{\alpha \neq \beta} < P(\alpha), P(\beta) >$$

$S(\Gamma)$ reflects the "tightness" of the document group, since $S(\Gamma)$ is higher if the documents of Γ are more similar to one another.

4.3 Group Average Clustering [4]

4.3.1 basic principles. Let G be a set of disjoint document groups (*clusters*). The group average clustering method finds the two disjoint clusters Γ and Δ which maximize the *average pair-wise similarity* $S(\Gamma \cup \Delta)$ over all other choices from G , and merges them to form a new cluster $(\Gamma \cup \Delta)$. A new set of disjoint clusters *G is derived by *G = (G – $\{\Gamma, \Delta\}$)$\cup$ $\{\Gamma \cup \Delta\}$.

For a cluster, the average pairwise similarity $S(\Gamma)$ is related to the inner product $<\mathbf{P}(\Gamma), \mathbf{P}(\Gamma)>$ by the following:

$$<P(\Gamma),P(\Gamma)> = <\sum_{\alpha\in\Gamma} P(\alpha), \sum_{\beta\in\Gamma} P(\beta)> = \sum_{\alpha\in\Gamma}\sum_{\beta\in\Gamma} <P(\alpha),P(\beta)>$$

$$= \sum_{\substack{\alpha\in\Gamma \\ \beta\in\Gamma}}\sum_{\beta\neq\alpha} <P(\alpha),P(\beta)> + \sum_{\alpha} <P(\alpha),P(\alpha)>$$

$$= |\Gamma|(|\Gamma|-1)S(\Gamma) + |\Gamma|,$$

hence

$$S(\Gamma) = \frac{<P(\Gamma),P(\Gamma)> - |\Gamma|}{|\Gamma|(|\Gamma|-1)}.$$

If two disjoint clusters Γ and Δ are merged to form a new cluster Λ, then we have :

$\Lambda = \Gamma \cup \Delta$, $|\Lambda| = |\Gamma| + |\Delta|$,

profile of $\Lambda = \mathbf{P}(\Lambda)$

$$= \sum_{\lambda\in\Lambda} P(\lambda) = \sum_{\lambda\in\Gamma} P(\lambda) + \sum_{\lambda\in\Delta} P(\lambda)$$

$$= \mathbf{P}(\Gamma) + \mathbf{P}(\Delta),$$

$$< \mathbf{P}(\Lambda), \mathbf{P}(\Lambda) > = < \mathbf{P}(\Gamma) + \mathbf{P}(\Delta), \mathbf{P}(\Gamma) + \mathbf{P}(\Delta) >$$

$$= < \mathbf{P}(\Gamma), \mathbf{P}(\Gamma) > + 2 < \mathbf{P}(\Gamma), \mathbf{P}(\Delta) > + < \mathbf{P}(\Delta), \mathbf{P}(\Delta) >,$$

,and

$$S(\Lambda) = \frac{< \mathbf{P}(\Lambda), \mathbf{P}(\Lambda) > - (|\Gamma| + |\Delta|)}{(|\Gamma| + |\Delta|)((|\Gamma| + |\Delta|) - 1)}$$

4.3.2 clustering procedure revisited. With the above background, we can have a detailed discussion of steps 3,4 and 5 of the clustering procedure mentioned in section 3.4.

4.3.2.1 Step 3: constructing initial cluster similarity matrix. Initially, there are N distinct documents, each constituting a singleton cluster. For example, document α_n constitutes cluster $S_n = \{\alpha_n\}$ for n ranging from 1 to N. A N by N matrix is constructed to measure their similarity. Specifically, the matrix element at the interaction of i-th row and j-th column, denoted by M_{ij}, contains a pair (S_{ij}, P_{ij}). That is, $M_{ij} = (S_{ij}, P_{ij})$, where

$S_{ij} = S(\Lambda)$ = *average pairwise similarity* for a new cluster generated from merging two original clusters S_i and S_j ; i.e. , $S(\Lambda) = S(S_i \cup S_j)$, and when i=j , $|\Lambda| = 1$, $S_{ij} = S(\Lambda)$ is undefined. In this case, assign 0 to S_{ij}.

P_{ij} = inner product of the profile for cluster Λ $= < \mathbf{P}(\Lambda), \mathbf{P}(\Lambda) >$, with $\Lambda = S_i \cup S_j$

4.3.2.2 results from previous iteration. Step 4 and step 5 are done iteratively. For a general treatment, assume that after some iteration, we have u clusters : Γ_1, Γ_2 ,... Γ_i ,... Γ_j , ... Γ_u . The similarity matrix has element $M_{ij} = (S_{ij}, P_{ij})$ for i and j ranging from 1 to u, where

$S_{ij} = S(\Lambda)$ = *average pairwise similarity* for a new cluster generated from merging two clusters Γ_i and Γ_j ; that is , $S_{ij} = S(\Lambda) = S(\Gamma_i \cup \Gamma_j)$, and

P_{ij} = inner product of the profile for the new cluster Λ $= < \mathbf{P}(\Lambda), \mathbf{P}(\Lambda) >$ $= < \mathbf{P}(\Gamma_i \cup \Gamma_j), \mathbf{P}(\Gamma_i \cup \Gamma_j) >$

4.3.2.3 step 4 : merging the two disjoint clusters that will maximize the average pairwise similarity when merged to form a new cluster. Assume clusters Γ_s and Γ_t (let $1 \leqq s < t \leqq u$ for convenience) satisfy the merging condition described in section 4.3.1 and are combined to form a new cluster $*\Gamma_s = \Gamma_s \cup \Gamma_t$. Now, the t-th row and the t-th column of the similarity matrix should be marked as invalid because Γ_t has been absorbed into the new cluster $*\Gamma_s$.

4.3.2.4 step 5: updating the similarity matrix. The s-th row and the s-th column of the similarity matrix must be updated, because, for the matrix elements $M_{sj} = (S_{sj}, P_{sj})$ (and $M_{js} = (S_{js}, P_{js})$ by symmetry) in these locations, S_{sj} no longer represents the average pair-wise similarity $S(\Gamma_s \cup \Gamma_j)$ of the previous iteration, but rather the new average pair-wise similarity $S(*\Gamma_s \cup \Gamma_j)$ for all j with $1 \leqq j \leqq u$ (Note that $j \neq t$, however, since t-th row and t-th column has been marked invalid in step 4.). To emphasize this new iteration, M_{sj} is denoted as $*M_{sj} = (*S_{sj}, *P_{sj})$.

This updating, although complicated, can take advantage of the results of the previous iteration to reduce the required computations. These available results are (refer to section **4.3.2.2**) the following inner products :

1. $P_{sj} = < \mathbf{P}(\Gamma_s \cup \Gamma_j), \mathbf{P}(\Gamma_s \cup \Gamma_j) >$
2. $P_{st} = < \mathbf{P}(\Gamma_s \cup \Gamma_t), \mathbf{P}(\Gamma_s \cup \Gamma_t) >$
3. $P_{tj} = < \mathbf{P}(\Gamma_t \cup \Gamma_j), \mathbf{P}(\Gamma_t \cup \Gamma_j) >$
4. $P_{ss} = < \mathbf{P}(\Gamma_s \cup \Gamma_s), \mathbf{P}(\Gamma_s \cup \Gamma_s) > = < \mathbf{P}(\Gamma_s), \mathbf{P}(\Gamma_s) >$
5. $P_{tt} = < \mathbf{P}(\Gamma_t \cup \Gamma_t), \mathbf{P}(\Gamma_t \cup \Gamma_t) > = < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_t) >$
6. $P_{jj} = < \mathbf{P}(\Gamma_j \cup \Gamma_j), \mathbf{P}(\Gamma_j \cup \Gamma_j) > = < \mathbf{P}(\Gamma_j), \mathbf{P}(\Gamma_j) >$

The new matrix element $*M_{sj} = (*S_{sj}, *P_{sj})$ ($*M_{js} = (*S_{js}, *P_{js})$ can be similarly derived) can be determined with the above six items for $1 \leqq j \leqq u$, with $j \neq s, t$:

$$*P_{sj} = < P(*\Gamma_s \cup \Gamma_j), P(*\Gamma_s \cup \Gamma_j) > \quad (*\Gamma_s = \Gamma_s \cup \Gamma_t)$$
$$= < \mathbf{P}(*\Gamma_s), \mathbf{P}(*\Gamma_s) > + 2 < \mathbf{P}(*\Gamma_s), \mathbf{P}(\Gamma_j) > + < \mathbf{P}(\Gamma_j), \mathbf{P}(\Gamma_j) >$$
$$= < \mathbf{P}(\Gamma_s \cup \Gamma_t), \mathbf{P}(\Gamma_s \cup \Gamma_t) > + 2 < \mathbf{P}(\Gamma_s \cup \Gamma_t), \mathbf{P}(\Gamma_j) > + P_{jj}$$
$$= P_{st} + 2 < \mathbf{P}(\Gamma_s \cup \Gamma_t), \mathbf{P}(\Gamma_j) > + P_{jj}$$
$$= P_{st} + P_{jj} + 2 \{ < \mathbf{P}(\Gamma_s), \mathbf{P}(\Gamma_j) > + < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_j) > \}$$
$$= P_{st} + P_{jj} + \{ 2 < \mathbf{P}(\Gamma_s), \mathbf{P}(\Gamma_j) > \} + 2 < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_j) >$$
$$= P_{st} + P_{jj} + 2 < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_j) > + \{ < \mathbf{P}(\Gamma_s \cup \Gamma_j), \mathbf{P}(\Gamma_s \cup \Gamma_j) > - < \mathbf{P}(\Gamma_s), \mathbf{P}(\Gamma_s) > - < \mathbf{P}(\Gamma_j), \mathbf{P}(\Gamma_j) > \}$$

($\because$ vector equation $\{2A\ B = (A+B)\ (A+B) - A\ A - B\ B\}$, and for disjoint clusters Γ_s and Γ_j, $\mathbf{P}(\Gamma_s \cup \Gamma_j) = \mathbf{P}(\Gamma_s) + \mathbf{P}(\Gamma_j)$)

$$= P_{st} + P_{tj} + [\, 2 < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_j) >] + \{ P_{sj} - P_{ss} - P_{jj} \}$$
$$= P_{st} + P_{tj} + \{ P_{sj} - P_{ss} - P_{jj} \}$$
$$+ [< \mathbf{P}(\Gamma_t \cup \Gamma_j), \mathbf{P}(\Gamma_t \cup \Gamma_j) > - < \mathbf{P}(\Gamma_t), \mathbf{P}(\Gamma_t) > - < \mathbf{P}(\Gamma_j), \mathbf{P}(\Gamma_j) >]$$
$$= P_{st} + P_{jj} + \{ P_{sj} - P_{ss} - P_{jj} \} + [P_{tj} - P_{tt} - P_{jj}]$$
$$= (P_{st} + P_{sj} + P_{tj}) - (P_{tt} + P_{jj} + P_{ss})$$

Note that the shaded items are the available results from previous iteration.

$$*S_{sj} = S(*\Gamma_s \cup \Gamma_j) \quad , \text{with } *\Gamma_s = \Gamma_s \cup \Gamma_t$$
$$= [< P(*\Gamma_s \cup \Gamma_j), P(*\Gamma_s \cup \Gamma_j) > - (|*\Gamma_s| + |\Gamma_j|)] / (|*\Gamma_s| + |\Gamma_j|) (|*\Gamma_s| + |\Gamma_j| - 1)$$

($\because$ the formula in section 4.3.1)

$$= [*P_{sj} - (|\Gamma_s| + |\Gamma_t| + |\Gamma_j|)] / [(|\Gamma_s| + |\Gamma_t| + |\Gamma_j|) (|\Gamma_s| + |\Gamma_t| + |\Gamma_j| - 1)]$$

Note that the new $*P_{sj}$ above has been obtained in terms of results of the previous iteration.

For j=s, we have

$$*P_{sj} = *P_{ss} = < \mathbf{P}(*\Gamma_s), \mathbf{P}(*\Gamma_s) > = < \mathbf{P}(\Gamma_s \cup \Gamma_t), \mathbf{P}(\Gamma_s \cup \Gamma_t) > = P_{st}$$

and

$$*Ssj = *Sss = S(*\Gamma s) = \frac{*Psj - (|\Gamma s| + |\Gamma t|)}{(|\Gamma s| + |\Gamma t|)((|\Gamma s| + |\Gamma t|) - 1)}$$

Note that the new $*P_{sj}$ above has been obtained in terms of results of the previous iteration. Hence, the meanings of the pair (S_{ij}, P_{ij}) of the similarity matrix are clear: S_{ij} is used to choose the most similar clusters for merging, and P_{ij} is used to speed up the updating of similarity matrix for each iteration.

5. Performance Analysis

5.1 Multi-Search

The most obvious advantage of Multi-Search is the stable throughput of documents; i.e., the documents come out at a stable rate, which is shown in Figure 5.1.

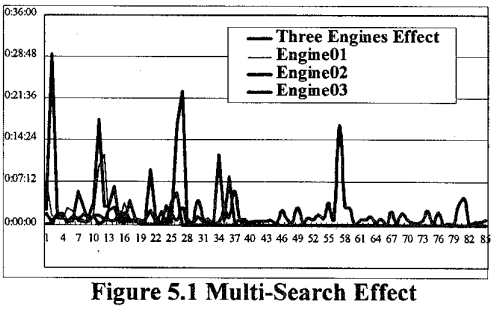

Figure 5.1 Multi-Search Effect

(Query = Classification)

The x-axis is the number of retrieved documents, and the y-axis is the time lapse between successive documents. The curve is rough for a single search engine, but the aggregate curve for multi-search mechanism is smoothed when all engines are working in parallel. However, it should be noted that when more individual engines complete their jobs, the aggregate curve for multi-search is determined by fewer remaining engines and is therefore less smooth.

5.2 Clustering

The clustering method used in this system is *Group Average Link* algorithm [2]. Figure 5.2 shows the clustering result from this algorithm compared with two other algorithms : *Single Link* and *Complete Link*. Single Link algorithm, though simple and efficient, has a tendency to form long straggly clusters known as chaining effect. Complete Link algorithm, on the other extreme, usually produces small, tightly bound clusters. The *Group Average Link* algorithm, despite its complexity, results in an acceptable structure between those of the other two methods.

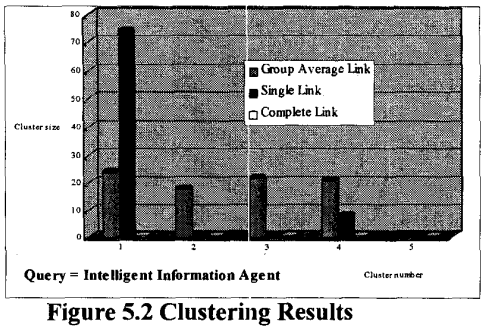

Figure 5.2 Clustering Results

(query = Intelligent Information Agent)

6. Conclusions and Future Works

This paper discussed three strategies for constructing a personal digital library: Multi-Search, Classification and User Profile. The Multi-Search method not only broadens the domains of retrieved information but also produces a stable and higher throughput of documents. Classification merges and organizes the documents from various information sources fetched by Multi-Search, and makes it easier for the user to identify the class of relevant documents. User Profile keeps track of the interest and traits of the user's information need to streamline information search, and helps the user construct a customized category. Moreover, this paper derived in detail the techniques for speeding up the iterative process for clustering.

Personal Digital Library is an active area of research, and our work is just a beginning for a more versatile and effective system. Listed below are but a few possible future directions of research:

1. Artificial Intelligence (AI) technique such as natural language processing can be used to make the query more natural and friendly.
2. Highly graphic user interface and information visualization can make the interaction with the system more intuitive for the average users.
3. The theory of cognitive science may be utilized for information retrieval.
4. As multimedia data become popular in WWW, the techniques for querying and retrieving composite documents are worth studying.

References

[1] C.H. Chang and C.C. Hsu, "Customizable Multi-engine Search Tool with Clustering" , Computer Networks and ISDN System 29, 1997, pp.1217-1224

[2] R.M. Cormack, " A Review of Classification" , Journal of the Royal Statistical Society, Ser. A. 134, pp.321-353

[3] D.D. Cutting, D.R. Karger, Jan O. Pedersen and John W. Tukey, " Scatter/ Gather: A Cluster-based Approach to Browsing Large Document Collections", ACM SIGIR 92'

[4] W.B. Frakes and R. Baeza-Yates edt, Information Retrieval: Data Structure and Algorithms, Prentice-Hall, 1992

[5] Robert R. Korfhage, Information Storage and Retrieval, Wiley, 1997

[6] B. Pinkerton, "Finding What People Want : Experiences with the WebCrawler" , 2nd Int. WWW Conference '94, July 1994

[7] H.C. Romesburg , *Cluster Analysis for Researchers,* Wadsworth, Belmont, CA. 1984

[8] E. Selberg and O. Etzioni, "Multi-engine Search and Comparison Using the MetaCrawler", Proc. 4th World Wide Web Conference, Dec. '95

[9] B. Yuwono and D. Lee, "Search and Ranking Algorithms for Locating Resources on the World Wide Web", Proc. 12th Int. Conference on Data Engineering, New Orleans, LA. , 1996