Multilayer SOM With Tree-Structured Data for Efficient Document Retrieval and Plagiarism Detection

Tommy W. S. Chow and M. K. M. Rahman

Abstract—**This paper proposes a new document retrieval (DR) and plagiarism detection (PD) system using multilayer self-organizing map (MLSOM). A document is modeled by a rich tree-structured representation, and a SOM-based system is used as a computationally effective solution. Instead of relying on keywords/lines, the proposed scheme compares a full document as a query for performing retrieval and PD. The tree-structured representation hierarchically includes document features as document, pages, and paragraphs. Thus, it can reflect underlying context that is difficult to acquire from the currently used word-frequency information. We show that the tree-structured data is effective for DR and PD. To handle tree-structured representation in an efficient way, we use an MLSOM algorithm, which was previously developed by the authors for the application of image retrieval. In this study, it serves as an effective clustering algorithm. Using the MLSOM, local matching techniques are developed for comparing text documents. Two novel MLSOM-based PD methods are proposed. Detailed simulations are conducted and the experimental results corroborate that the proposed approach is computationally efficient and accurate for DR and PD.**

Index Terms—**Document retrieval (DR), multilayer self-organizing map (MLSOM), plagiarism detection (PD), tree-structured representation.**

I. Introduction

The easy access to the Internet has made disseminating knowledge across the world much easier. Documents can easily be searched, copied, saved, and reused for different purposes. Document retrieval (DR) and categorization has also become important in facilitating the handling of large number of documents. As a result, plagiarism detection (PD) has become increasingly important when more information can be electronically exchanged, reused, and copied. DR refers to finding similar documents to a given query. Document categorization is another text-related application, where a given document is required to be labeled with a class or classes with an automated system to reduce human labor. Categorization task can be performed by machine learning tool such as support vector machine (SVM) [1]. SVM performs the classification task by constructing hyperplanes in a multidimensional space that separates different classes [2]. The extracted class information of documents can be utilized in DR system. Other than matching text content, DR also uses rank information of documents such as Google's PageRank. Ranking of document can be performed with SVM-based interactive retrieval systems that learn from users' relevance feedback [3]. In [4], the problem of DR is formulated as binary classification using SVM, where document collection is classified into relevant or nonrelevant; the relevant documents are then returned in the retrieval result. Much research work on DR has been reported in the literature but further improvement in the way of speeding up the retrieval time and increasing the accuracy has always been welcome. On the other hand, plagiarism is the act of copying text from one's work without permission or proper acknowledgement. Research work on PD is scarce. Currently, there are several approaches of conducting document searching. Most of the traditional methods are "keywords"-based searching, e.g., Google, which relies on the keywords provided by users. The search engine usually provides ranked documents mostly based on the occurrence of the keywords. Searching similar documents for a query document is another type of DR. It is flexible to users, because a query document may consist of few keywords, paragraphs, and so on. Thus, retrieval by a query document is powerful but computationally complex compared with the "keyword-based searching" method. In addition to retrieval, there are search engines, e.g., Vivisimo, that are able to cluster retrieved documents [5] into different groups. While DR facilitates knowledge sharing immensely, it makes partly or fully copying of others' work into one's documents easy. PD [6]–[9] is a research area that is closely related to DR. With the immense growth in document collections, PD is difficult because it is not easy to develop an effective and computationally efficient method that can model a document accurately. Currently, most models use crude document features, but they are usually unable to detail a document description. Thus, there is a need to improve document-feature representation in order to improve the performance of DR and PD.

Traditional DR methods rely on term-based related models [10]–[14] such as Boolean, probabilistic, language, and vector space model (VSM). Boolean model uses Boolean operators on query words. It is simple but unable to provide any ranking of the retrieved documents. Probabilistic model uses the idea of assigning a relevance score to each term of a document, whereas the score is related to the occurrence of that term in some known

Manuscript received June 26, 2008; revised October 15, 2008 and February 02, 2009; accepted April 21, 2009. First published July 28, 2009; current version published September 02, 2009. This work was supported by the City University of Hong Kong under Grant 7001998-570.

T. W. S. Chow is with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong (e-mail: eetchow@cityu.edu.hk).

M. K. M. Rahman was with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong. He is now with the Department of Electrical and Electronic Engineering, United International University, Dhaka 1209, Bangladesh (e-mail: masuk@eee.uiu.ac.bd).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2009.2023394

relevant documents. Apart from these models, language model [14], a statistical method of ranking of documents, has become quite popular. VSM model, the most popular and widely used compared with others models, counts the frequency of each term of a *vocabulary* for a given document, where *vocabulary* is a list of terms/words used for feature description. A term weight vector is then constructed using this "term frequency" together with "document frequency," where document frequency is the number of documents where the term appears. Similarity between two documents is performed by using "cosine" measure or any other distance function [11]. VSM model thus requires a lengthy vector, because the number of words involved is usually huge. This causes a significant increase in computational burden making the direct use of VSM model impractical for handling a large data set. To overcome this limitation, latent semantic indexing (LSI) [15] was developed to compress the feature vector of the VSM model into low dimension. In addition to feature compression, LSI model is useful in encoding the semantics [16], [17]. Besides the LSI, self-organizing map (SOM) was employed for document-feature projection [18], [19] and dimensionality reduction. In [20], LSI is used to encode the semantic concept to some extent. In [21], it was suggested to use information extraction (IE) to improve DR systems performance. According to [22], DR is to retrieve relevant documents from database, and IE is to extract relevant information from the retrieved documents. IE has been developed through a series of Message Understanding Conferences [22]. Other content such as image has also been included to enhance the retrieval system [20]. In [23], thesaurus is added to widen the searching space by incorporating similar words to the query words. It is useful in some extent because a semantic context can be represented by different sets of words. It shows that using domain specific thesaurus is more effective than using generic thesaurus on its own.

It is worth noting that all the above described methods use *flat feature* representation. In fact, *flat feature* representations are a crude representation of a document. For example, two documents containing similar term frequencies may be of different contextually when the spatial distribution of terms is very different, i.e., *school, computer*, and *science* mean different things when they appear in different parts of a page compared to *school of computer science* that appear together. Thus, it is important to account contextual similarity based on the ways that words are used throughout the document, i.e., sections and paragraphs. But, until now, most document models incorporate only term frequency and do not include such contextual information. Tree structure, a rich data representation, is found to be effective in including the spatial information. Details of tree structure will be elaborated on in later section of this paper.

Speeding up the retrieval operation is also an equally important issue. In [24], it was suggested that clustering of large document data set could be useful, because it narrows the searching scope by comparing a query to a group of documents that are clustered according to their nature. The searching effort is reduced compared with the mechanism of searching the whole database. Fuzzy concept [25] together with clustering technique [26] is also used in DR. Other than clustering, researchers also proposed a new file system [27] to speed up the retrieval process.

SOM is a versatile unsupervised neural network used for generating a topologically ordered map and clusters. Based on a topologically ordered map of documents, SOM facilitates users to find similar documents that are close to each other on the SOM map. SOM is computationally efficient in performing fast DR [28], because it only compares a query document with neurons instead of all documents in the database. There are variants of SOM, e.g., WEBSOM [18] and latent semantic indexing and SOM (LSISOM) [15], that are used for document visualization and interactive browsing. Another interesting application of SOM for hierarchical document organization and browsing can be found in [5] and [29]. The topological organization of content (TOC) method [5], which can be seen as hierarchical clustering, spans a set of 1-D growing SOMs. It is computationally effective and scalable for large data set. Despite all the successes, all these approaches rely on typical term-frequency information. In order to encode a document in a better way, we propose to use tree-structured representation. Tree structure is a powerful approach that has been successfully applied to numerous applications of pattern recognition [30]–[34]. We developed multilayer SOM (MLSOM) [35] for handling unsupervised learning of tree-structured data, which was used in clustering, visualization, retrieval, and classification [32], [35]. In this paper, we use MLSOM for handling DR and PD.

Most traditional DRs are performed using the global characteristics of a document, i.e., overall term-frequency information. On the other hand, copying a small part of text is regarded as plagiarism, although it does not make the "plagiarized document" globally more similar to the "source document." There is a need to analyze two documents locally for detecting plagiarism in contrast with global similarity analysis in the retrieval process. Plagiarism can be of different types in nature, from copying text to copying idea [36]. The scope of our present work focuses on copying text with or without minor modification of the words and grammar. According to [36], there exist three different broad categories of detecting textual plagiarism: 1) comparing the query over a database [7]–[9], 2) comparing core sentences/phrases through a search engine [37], and 3) comparison by stylometric analysis [38]. Most methods fall within the first category where detection is performed by dividing documents into small blocks such as a paragraph [9], a sentence [7], or a word [8]. The matching between two documents is carried out based on the overlapping of those blocks. These blocks are registered against a hash table. Thus, all documents of a database are hashed in a modular form. A query document is similarly divided into small blocks that are subsequently matched in the hash table for PD. In fact, most of the existing systems fail to detect plagiarism by paraphrasing (changing words of a copied paragraph) [36], because they do not account the overlap when plagiarism is done by changing words and structure of sentences. Overlap measure can be carried out in different ways. In [39], a method based on linguistically motivated plagiarism patterns was proposed that allows partial match when sentences are made different by changing words or structure. To scale up PD for a large database, a string-matching fingerprint-based system [40] was developed. It calculates the document fingerprint from character level instead of words. The process is sped up but at the expense of losing semantic information. Another string-matching algorithm can be found in [6]. It uses suffix trees to detect plagiarism at an enhanced speed. It requires less space, but suffix trees [6] do not scale well and the algorithm is only applicable to a small data set. Most of these approaches are only suitable for a word-by-word matching over a limited document collection. According to [9], most existing copy detection mechanisms employ an exhaustive sentence-based comparison to match a potential plagiarized document with all the registered documents. This approach is not scalable due to the potentially large number of registered documents and the large number of sentences in each document. Also the security level of the existing mechanisms is quite weak; a plagiarized document could easily bypass the detection mechanism by performing a minor modification on each sentence. In [9], a tree-structured document feature was proposed and the method was further improved in [41]. But the method uses a nonnumeric feature representation making it unable to utilize SOM or other clustering technique to speed up the handling of a large database.

This paper proposes a novel approach using a three-level tree representation and MLSOM. Document contents are hierarchically organized into different layers of a SOM, i.e., paragraphs are organized at the bottom layer, pages are organized at the middle layer, and the whole documents are organized at the top layer. Different layers play different roles; the top layer performs document clustering, and the other layers compress local features. The global and local document characteristics organized at the top and bottom layers are essential for DR and PD, respectively. The system also supports a fast relevance feedback that improves the retrieval accuracy significantly.

In summary, the contribution of this paper is twofold. First, we propose a tree-structured document model for DR and PD. The model enables global and local characteristics of a document to be included in a hierarchical way. Thus, it enhances DR accuracy and enables PD. Second, we present an efficient system for both DR and PD, where MLSOM [35] serves as a computationally efficient clustering algorithm. The method can serve as a unified platform for performing both DR and PD. Also, it should be noted that the described DR mechanism does not rely on comparing keywords/lines; it relies on comparing the text of the whole document. The rest of this paper is organized as follows. Section II presents the details of document feature representation and feature extraction. Section III provides the details of the MLSOM training. Section IV describes how DR and PD are performed. Experimental results are presented and analyzed in Section V. The conclusion is finally drawn in Section VI.

II. Structured Representation of Document Features

To extract the tree structure, a document is partitioned into pages that are further partitioned into paragraphs. We have developed a Java code to perform such segmentation, and have considered only "html" documents at this stage. In "html" format document, paragraphs can be easily identified using html tags. First, a document is segmented into a number of paragraphs. The paragraphs are then merged into pages using a minimum threshold value for the number of words on a page. It is noted that any text appearing within the html tags, which are used for formatting, is not accounted for word count or document feature extraction.

The document partitioning process can be summarized as follows.

1) Partition the document into paragraph blocks using the html paragraph tag "⟨p⟩" and new line tag "⟨br⟩."
2) Merge the subsequent paragraph blocks to form a new page until the total number of words of the merged blocks exceeds a page threshold value 1000. There is no minimum threshold for the last page. The page blocks are formed.
3) Now each page is split into a smaller blocks using more html tags: "⟨p⟩," "⟨br⟩," "⟨il⟩," "⟨td⟩," etc. Merge these subsequent blocks in the same fashion as in step 2) to form a new paragraph until total number of words of the merged blocks exceeds a page threshold value 100. The minimum threshold for the last paragraph of a page is kept at 40; otherwise the paragraph is merged with the previous paragraph.

In html documents, there is neither definite page boundary/break nor any rule for the minimum/maximum number of words for paragraphs/pages. But the use of a threshold of a word count enables us to form a hierarchical structure containing the characteristics from global to local. The contents are structured in a way of "document $\rightarrow$ pages $\rightarrow$ paragraph" tree. This is an effective way of generating a tree structure, and it can be further improved by using a form of "document $\rightarrow$ sections $\rightarrow$ paragraphs" or "document $\rightarrow$ sections $\rightarrow$ sentences," but it demands more complex algorithm for partitioning a document.Fig. 1(d) illustrates the three-level tree-structured representation of a document. The root node at the first level represents the whole document, the second level nodes represent different pages, and the third level nodes represent paragraphs of the pages. Nodes contain compressed features describing frequency distribution of different words. Nodes at different levels contain the same word-frequency features, but they are extracted from different contents of the document. Two documents having similar word histograms at root nodes can be completely different in terms of semantics/context, because different spatial distributions of the same set of words can result in different meaning/context. This is what is reflected by the discriminative lower parts of the tree data (second/third level nodes).

Tree-structured data can be used effectively for both DR and PD. For DR, similarity between two documents can be measured using the whole tree, where root nodes give us global similarity and the third-level nodes provide local similarity. Tree structure can also provide information of plagiarism through the third-level nodes which encode the local spatial information of word distribution. To extract the features, word histograms are computed for the nodes at different levels. We then apply principal component analysis (PCA), a tool to project higher dimensional data into lower dimensional feature without losing much statistical information, to the word-histogram vector. Fig. 1(a)–(c) details the block diagrams for extracting tree-structured features. In preprocessing, word extraction, vocabulary construction, and generation of PCA projection matrix are performed. Then, a document is portioned into a three-level tree, and nodes' features are computed using the feature–extraction module shown in Fig. 1(a). The overall

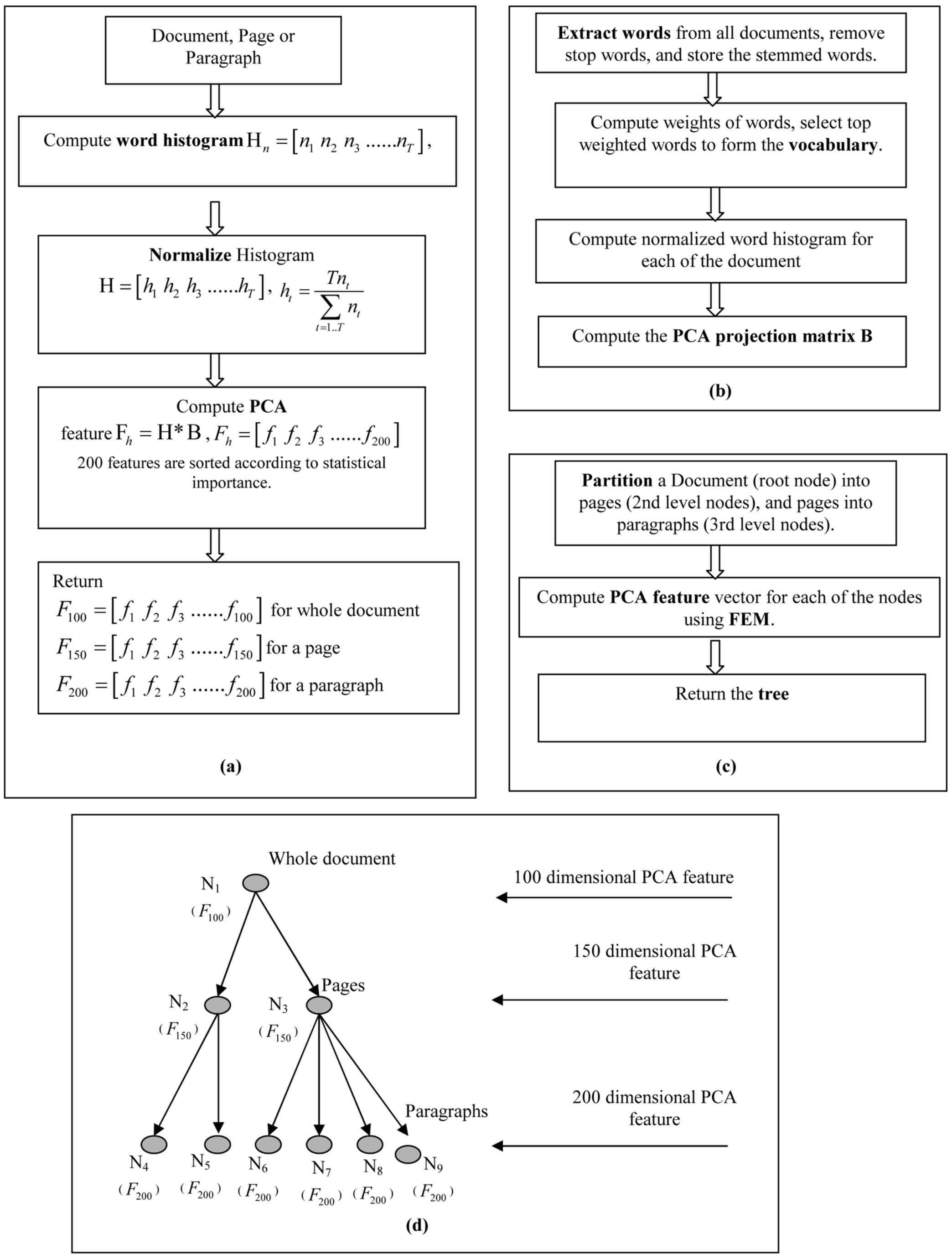

Document, Page or Paragraph
Compute word histogram $H_n = [n_1\ n_2\ n_3 \ldots\ldots n_T]$,
Normalize Histogram
$H = [h_1\ h_2\ h_3 \ldots\ldots h_T]$, $h_t = \frac{Tn_t}{\sum_{t=1..T} n_t}$
Compute PCA feature $F_h = H * B$, $F_h = [f_1\ f_2\ f_3 \ldots\ldots f_{200}]$
200 features are sorted according to statistical importance.
Return
$F_{100} = [f_1\ f_2\ f_3 \ldots\ldots f_{100}]$ for whole document
$F_{150} = [f_1\ f_2\ f_3 \ldots\ldots f_{150}]$ for a page
$F_{200} = [f_1\ f_2\ f_3 \ldots\ldots f_{200}]$ for a paragraph
(a)
Extract words from all documents, remove stop words, and store the stemmed words.
Compute weights of words, select top weighted words to form the vocabulary.
Compute normalized word histogram for each of the document
Compute the PCA projection matrix B
(b)
Partition a Document (root node) into pages (2nd level nodes), and pages into paragraphs (3rd level nodes).
Compute PCA feature vector for each of the nodes using FEM.
Return the tree
(c)
Whole document
N_1 (F_{100})
100 dimensional PCA feature
Pages
N_2 (F_{150})
N_3 (F_{150})
150 dimensional PCA feature
Paragraphs
200 dimensional PCA feature
N_4 (F_{200})
N_5 (F_{200})
N_6 (F_{200})
N_7 (F_{200})
N_8 (F_{200})
N_9 (F_{200})
(d)

Fig. 1. (a–c) Block diagrams of feature extraction procedures. (d) Document representation. (a) Feature extraction method (FEM). (b) Preprocessing. (c) Extraction of tree-structured feature. (d) Document representation by tree-structured feature.

procedure of extracting tree-structured feature can be summarized as follows.

1) Extract words from all the documents in a database and apply stemming to each word. Porter stemming algorithm [42] is used to extract stem of each word, and stems are used for feature extraction instead of original words. In this way, "work," "works," and "worked" are all considered being the same word. Remove the stop words (set of common words such as "the," "is," etc.). Store the stemmed words together with the information of term frequency f_T (the frequency of a word in all documents) and the document frequency f_d (the number of documents where a word appeared).

2) *Vocabulary* construction: Using term frequency f_t and document frequency f_d, we calculate the weight of each word that is similar to the term weighting in VSM [43]

$$W_{\text{term}} = \text{idf} \times \sqrt{f_t} \tag{1}$$

where the inverse document frequency $\text{idf} = \log_2 (N/f_d)$, and N is the total number of documents in the whole database. We sort the words by using weight value in descending order and select the first T words. Alternatively, one can select words having a weight value above a threshold. The choice of T depends on the database.

3) Partition the document into pages and pages into paragraphs. Construct the document tree. The root node contains the histogram of a whole document, the second-level nodes are used for pages, and the third-level nodes are used for paragraphs. Besides the histogram features, a set of information is saved for each node that describes how the nodes are related to each other. They include: 1) a node index, 2) the level in tree, 3) the parent node index, and 4) the child node index.
4) For each document, calculate word histograms $H_n^q\{= [n_1 \quad n_2 \quad \ldots \quad n_T]\}$ for paragraphs, where n_t is the number of times the corresponding word t appears in a paragraph. The histograms for the pages and the whole documents can be obtained by hierarchical relationship $\{H_n^D = \sum H_n^P \mid P \in D\}$ and $\{H_n^P = \sum H_n^q \mid q \in P\}$, where H_n^D and H_n^P are histograms corresponding to a document and a page, respectively. Finally, we normalize the histogram
$$H = [h_1 \quad h_2 \quad h_3 \quad \ldots \quad h_T], \qquad h_t = f(n_t) \tag{2}$$
where $f(n_t)$ is a normalization function. The histogram vector can be normalized in a number of ways to form the normalized histogram vector $[h_1 \quad h_2 \quad h_3 \quad \ldots \quad h_T]$
$$\text{Mean norm:} \quad h_t = \frac{T n_t}{\sum\limits_{t=1\ldots T} n_t} \tag{3}$$
$$\text{Max norm:} \quad h_t = \frac{n_t}{\max\limits_{t=1\ldots T}(n_t)} \tag{4}$$
$$\text{VSM norm:} \quad h_t = n_t \times \log\left(\frac{N}{(f_d)_t}\right). \tag{5}$$
5) Use the normalized histogram to construct the PCA projection matrix B. To save the computational burden, we construct the matrix B only at the root nodes. We have used the MATLAB tool [44] to compute the PCA projection matrix.
6) Project the node features (normalized histogram) into lower dimensional PCA feature by using PCA projection matrix [44]
$$F_h = H * B \tag{6}$$
where B is the projection matrix of dimension $T \times n_F$, and n_F is the dimension of the projected feature. It should be noted that the histogram features H for the first-, second-, and third-level nodes are extracted from a document, a page, and a paragraph, respectively. The projected features in $F_h\{= [f_1, f_2, f_3, \ldots, f_{n_F},]\}$ are ordered according to their statistical importance. In our study, n_F was set to 200, but we further reduced the number of features used for the root node and second-level nodes. Thus, the new dimensions of the PCA feature for the first-, second-, and third-level nodes are 100, 150, and 200, respectively.
7) Save the *vocabulary* base and the projection matrix for making features of a new query document. The tree-structured features of a query document are extracted in the same way [see feature extraction method (FEM) in Fig. 1(a)] but excluding steps 1), 2), and 5), because they are required to compute only once over the database. Thus, negligible computational time to segment a query document and compute PCA feature using the projection matrix is required.

III. SOM Training

A. *Self-Organizing Map*

A basic SOM consists of M neurons located on a regular low-dimensional grid that is usually in 2-D. The lattice of a 2-D grid is either hexagonal or rectangular. Each neuron i has a d-dimensional feature vector $w_i = [w_{i1}, \ldots, w_{id}]$. The SOM algorithm is iterative. During the training process, the neurons are updated in a way that their feature vectors finally become representative of different data clusters in an orderly fashion. The iterative SOM training algorithm can be stated as follows.

Step 1. Set iteration $t = 0$.

Step 2. Randomly select a sample data vector $x(t)$ from a training set.

Step 3. Compute the distances between $x(t)$ and all feature vectors. The winning neuron, denoted by c, is the neuron with the feature vector closest to $x(t)$
$$c = \arg\max_i (S(x(t), w_i)), \qquad i \in \{1, \ldots, M\}. \tag{7}$$
S is a similarity function to compute similarity between $x(t)$ and w_i.

Step 4. Update the winner neuron and its neighbor neurons. The weight-updating rule in the sequential SOM algorithm can be written as
$$w_i(\mathrm{t}+1) = \begin{cases} w_i(t) + \eta(t) h_{ic}(t)(x(t) - w_i(t)), & \forall\, i \in N_c \\ w_i(t), & \text{otherwise.} \end{cases} \tag{8}$$
$\eta(t)$ is the learning rate which decreases monotonically with iteration t
$$\eta(t) = \eta_0 \cdot \exp\left(-\alpha \cdot \frac{t}{\tau}\right) \tag{9}$$
where η_0 is the initial learning rate, and α is an exponential decay constant set to 3 throughout our experiments. N_c is a set of neighboring neurons of the winning neuron, and $h_{ic}(t)$ is the neighborhood kernel function that defines the closeness of a neighborhood neuron to the winning neuron c at position (x_c, y_c). The neighborhood function $h_{ic}(t)$ also decreases gradually during the learning process
$$h_{ic}(t) = \exp\left(-\frac{\left[(x_i - x_c)^2 + (y_i - y_c)^2\right]}{2\sigma^2(t)}\right) \tag{10}$$
where $\sigma(t)$ is the width of the neighborhood function that decreases with iteration
$$\sigma(t) = \eta_0 \cdot \exp\left(-\frac{t}{\tau} \cdot \log \sigma_0\right) \tag{11}$$
where σ_0 is the initial width and τ is a time constant set to the maximum number of iterations.

Step 5. Stop when the maximum iteration is reached, or set $t = t + 1$ and go to step 2.

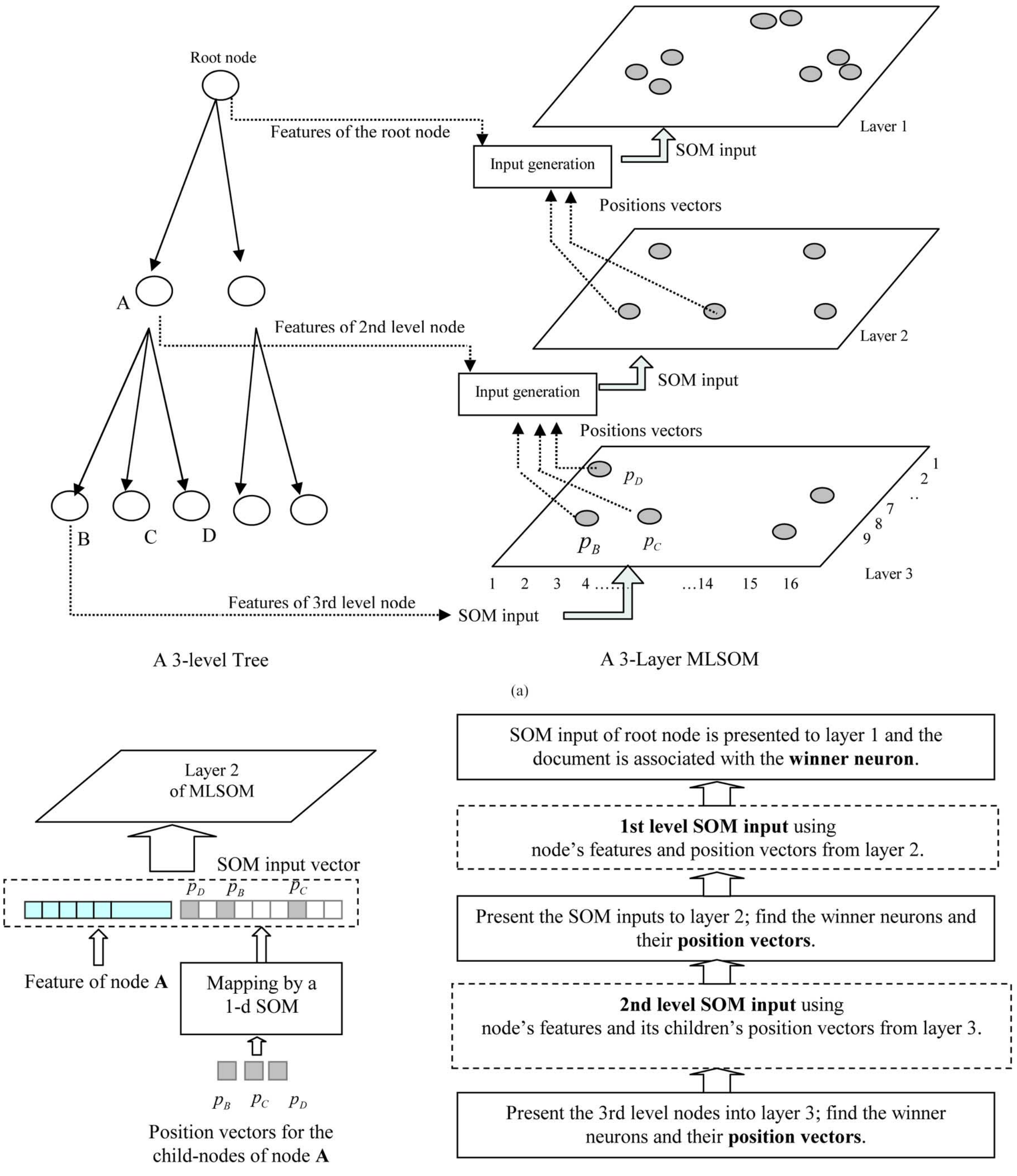

Root node
Features of the root node
Input generation
SOM input
Layer 1
Positions vectors
A
Features of 2nd level node
Layer 2
Input generation
SOM input
Positions vectors
p_D
p_B
p_C
B C D
1 2 3 4 … …14 15 16
1 2 .. 7 8 9
Layer 3
Features of 3rd level node
SOM input
A 3-level Tree
A 3-Layer MLSOM
(a)
Layer 2 of MLSOM
SOM input vector
p_D p_B p_C
Feature of node A
Mapping by a 1-d SOM
p_B p_C p_D
Position vectors for the child-nodes of node A
SOM input of root node is presented to layer 1 and the document is associated with the winner neuron.
1st level SOM input using node's features and position vectors from layer 2.
Present the SOM inputs to layer 2; find the winner neurons and their position vectors.
2nd level SOM input using node's features and its children's position vectors from layer 3.
Present the 3rd level nodes into layer 3; find the winner neurons and their position vectors.
(b)

Fig. 2. (a). Illustration of the mapping of a three-level tree-structured data into a three-layer MLSOM. (b) SOM input representation for the node A. (c) Block diagram illustrating mapping of a tree-structured data into MLSOM.

B. Multilayer SOM (MLSOM)

Conventional SOM cannot represent tree-structured data because the SOM input is only in a form of a flat vector. MLSOM [35], a kind of extended SOM model, was developed for processing tree-structured data. Its multilayered structure is specially designed to represent nodes of a tree data in a way of level-by-level. A tree data consists of nodes at different levels; a node consists of two types of information including the node features and its child nodes. In an MLSOM, there are as many SOM layers as the number of levels in the tree, i.e., three different SOMs for a three-layer MLSOM. Nodes of each level are processed by the corresponding layer of the MLSOM. Fig. 2(a) illustrates how a three level tree data is mapped into a three-layer MLSOM. First, the third-level nodes, containing features of different paragraphs, are mapped to the layer 3 SOM output. The winner neurons of different nodes or paragraphs (B, C, and D) are represented by their corresponding position vectors (p_B, p_C, and p_D). It is worth noting that at layer 3 there are huge numbers of child nodes representing the identity of all paragraphs. Using the layer 3 SOM, we are able to compress all the paragraphs into 2-D position vectors. Thus, the layer 2 SOM input vector consists of the second-level node (page) features and the position vectors. In this way, we can form a compressed input vector for layer 2 SOM. Fig. 2(b) depicts how the second-layer input vector is formed using the feature of a node and the position vectors p_B, p_C, and p_D. Fig. 2(b) also shows how the position vectors are mapped from the outputs of the layer 3 SOM into a structured format forming the input vector of the layer 2 SOM. At last, the root nodes are similarly processed at layer 1 SOM. As the root node represents the whole document tree, all documents are thus organized and associated with different winner neurons at the top layer SOM. It should be noted that the third- and second-layer SOMs serve as feature compression that compress the paragraph and page information, respectively. In this way, the whole tree can be compactly represented by the root node at the first-layer SOM. The whole procedures are summarized in the block diagram shown in Fig. 2(c).

In the tree data, a node at the kth level is represented by $X^k = [f_{1x}, f_{2x}, \ldots, f_{mx}, p_{1x}, p_{2x}, \ldots, p_{c_{\max}x}]^T$, where f_i represents the ith feature of node X^k, $p_{ix} (= [x_i, y_i]^T, x_i \in [0,1], y_i \in [0,1])$ is a normalized 2-D position vector of a neuron that compactly represents ith child node, and $c_{\max}$ is the maximum number of child nodes of the nodes at the kth level. The position vectors are mapped in the vector $[p_{1x}, p_{2x}, \ldots, p_{c_{\max}x}]$ according to spatial position that will be discussed later in this section. A node may have less than $c_{\max}$ number of child nodes, and some of p_{ix} may contain zero vector $[0,0]$. The weight vector W^k of a neuron at the kth layer is represented by $W^k = [f_{1w}, f_{2w}, \ldots, f_{mw}, p_{1w}, p_{2w}, \ldots, p_{c_{\max}w}]^T$, where f_{iw} is the ith weight and $p_{ix} (= [x_i, y_i]^T)$ is a 2-D vector. The following function computes the similarity between a node and a neuron in finding the winner neuron:

$$S(X,W) = C \cdot \frac{1}{m} \sum_{i=1}^{m} \{1 - |f_{ix} - f_{iw}|\} + \frac{1}{\sum_{j=1}^{c_{\max}} \mathbb{S}(p_j^x)} \sum_{j=1}^{c_{\max}} v_j \cdot \mathbb{S}(p_{jx}) \cdot \{1 - d(p_{jx}, p_{jw})\}$$

where

$$\mathbb{S}(p_{jx}) = \begin{cases} 1, & \text{if } p_{jx} \neq (0,0) \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

where C and v_j are weight parameters, and d is a Euclidean distance function. The first part of the expression computes the global similarity using the node feature, and the second part computes the local similarity using position vectors of child nodes. In the bottom-level nodes, only the first part of (12) is used. It was noted that v_j provides weighting among different child nodes. It is set to 1 by default, and is only modified during relevance feedback. In the first part of (12), C put relative emphasis between the global and local similarity measure. A larger value of C, $C > 1$, implies that emphasis is placed on global information, and a small value of C, $C < 1$, implies that local similarity is emphasized in comparing documents. The system provides flexibility to users to change the value of C to balance the similarity measure according to their expectation. The effect of C will also be discussed in later section of this paper.Fig. 3 illustrates the block diagrams of the MLSOM training. First, the third-layer SOM is trained with third-level nodes. Then, the inputs of the second-level nodes are formed by combining the nodes' features and position vectors of the corresponding child nodes [Fig. 2(b)]. Using these SOM inputs, the second-layer SOM is trained. Similarly, the top-layer SOM is trained by the SOM inputs for the root nodes. The whole training procedures are described as follows.

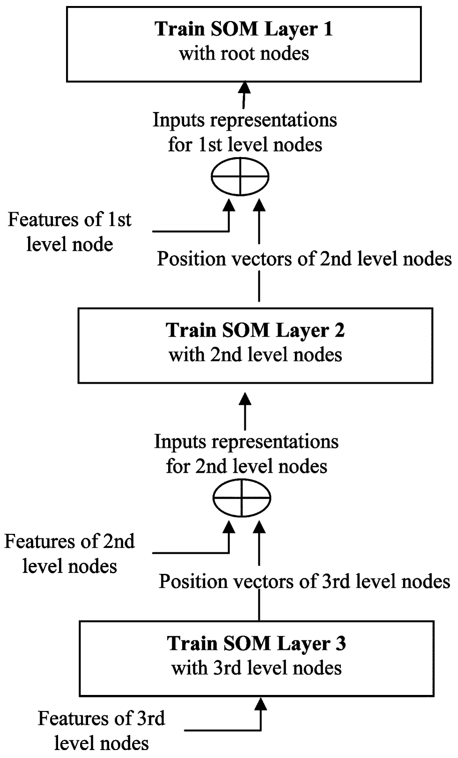


Fig. 3. Training sequences of MLSOM for the third-level tree-structured data.

Training Procedures of the MLSOM

```
(1) Initialize SOM for all layers
(2) **Loop** from the third to first layer
(3)     Select all the nodes of all tree data at the same level
        as the SOM layer
(4)     Create an input vector for each node by combining
        node's feature with mapped child nodes'
        position vectors.
(5)     **Loop** for training SOM
(6)         Randomly select an input vector
(7)         Find a winner neuron for the input vector on
            the current SOM layer
(8)         Update the current SOM layer
(9)     **End** of training loop
(10)    For each node, find the final winner neuron on the
        current SOM layer and save its position vector that
        will be used in the input of its parent node.
(11)    Unless it is the top layer, train a 1-D SOM using all
        position vectors on current level. This will be used to
        map position vectors in step (4).
(12) **End** of layer loop
```

In step (4), we map the position vectors of child nodes into the SOM input vector. The mapping of position vectors is conducted using a simple 1-D SOM that is trained in step (11). The mapping is required because it needs to make a meaningful matching between two sets of nodes using the second part of (12). It compares a child node of the first set with only one similar child node of another set. By performing this procedure, the two position sets can be compared by a speedy and meaningful "one-to-one" matching, instead of by a "many-to-many" matching [32]. The 1-D SOM is trained by all the position vectors ($p = \{x, y\}$) over the database at a particular level. After training the 1-D SOM, the following procedures are applied to map a set of position vectors.

Mapping of Position Vectors $\{p_1, p_2, \ldots p_c\}$ ***Using SOM of*** $c_{\max}$ ***Neurons***

```
Set $p_j^s \leftarrow [0,0]$, $N_j \leftarrow 0$ $j = 1 \ldots c_{\max}$
**Loop** for **each** $p_i$, $i = 1 \ldots c$
    Find three most matched neurons $j$, $k$, and $l$ for $p_i$
    If $N_j = 0$, $p_j^s \leftarrow p_i$ and $N_j \leftarrow N_j + 1$
    else if $N_k = 0$, $p_k^s \leftarrow p_i$ and $N_j \leftarrow N_k + 1$
    else if $N_l = 0$, $p_l^s \leftarrow p_i$ and $N_j \leftarrow N_k + 1$
    else $p_j^s \leftarrow \left(p_j^s \times N_j + p_i/N_j + 1\right)$ and $N_j \leftarrow N_j + 1$
    end if
**End**
**Return** $\{p_j^s\}$
```

The above procedures differ from [32] in a way that two or more position vectors in these procedures can be merged together by averaging. Thus, $c_{\max}$ is set to a value that is lower than the actual maximum number of child nodes. The length of the input vector that represents child nodes is then reduced, which results in a reduction of the dimension of SOM weight vector. The decrease of $c_{\max}$ is useful when the maximum number of child nodes in the database is large. The computational complexity of conducting one epoch of MLSOM training is $O(\sum_{k=1}^{3} N_k m_k (n_k + 2c_k))$, where N_k is the number of nodes at the kth level of trees, m_k is the number of neurons at the kth SOM layer, n_k is the number of input features of a node at the kth level, and c_k is the maximum number of children nodes of all the trees at the kth level. Also, it must be noted that the computational cost of the 1-D SOM is negligibly small.

IV. MLSOM-Based Document Retrieval and Plagiarism Detection

A. Preprocessing

The preprocessing for DR can be summarized as follows.

MLSOM Preprocessing

1) Train the MLSOM with all the tree-structured data of the database.
2) For the top-layer SOM, save the index of document data against their activated neurons. This will be used in the DR.
3) For the third-layer SOM, save the index of documents together with their node index (in the tree structure) against their activated neurons. This will be used in the PD. No preprocessing is required for the second-layer SOM.
4) Save the weight matrix of the MLSOM network and 1-D SOM used for mapping. Save the root node inputs of all documents constructed during training, which will be used for relevance feedback. Thus, the MLSOM, together with previously saved vocabulary base and PCA projection matrix, is ready to perform retrieval operation.

B. Document Retrieval and Relevance Feedback

A document is overall represented by the root node of its tree structure; the root nodes are processed at the top-layer SOM. Each document is indexed against its winner neuron at the top layer. This document association, which has been carried out in the preprocessing stage, is used for performing retrieval. The MLSOM-based retrieval system is summarized as follows.

MLSOM Retrieval

1) For a given query document, extract its tree structure. Calculate node features using the prestored vocabulary base and PCA projection matrix.
2) Match the nodes of the tree level-by-level from the bottom layer to the top layer, and find the most matched neurons on the top layer.
3) Go through the sorted neurons in descending order and append their associated documents into the retrieval list until at least N documents are found. N is the number of documents to be retrieved and is defined by users. Upon including documents from the last neuron, total number of documents can be higher than N.
4) Sort the documents of retrieved list according to query through comparing root node inputs by (12) and return the first N documents to users. This sorting helps to deliver better ranking in the retrieval results in terms of relevancy.

To provide additional support of relevance feedback, the system interface allows users to browse through documents. Users can select documents, which are more relevant to their target, as feedback to the system. The relevance feedback uses both query modification and modifying weight. First, the weight parameter v_j is modified as follows:

$$v'_j = \sum_{r=1}^{R} \mathbb{S}(\mathbf{p}_j^r) - \sum_{nr=1}^{NR} \mathbb{S}(\mathbf{p}_j^{nr}) \quad (13)$$

$$v''_j = v'_j - \min_{j=1\ldots c_{\max}} (v'_j) + 1 \quad (14)$$

$$v_j = \frac{c_{\max}}{\sum_{j=1}^{c_{\max}} v''_j} \cdot v''_j \quad (15)$$

where R and NR are the number of relevant and irrelevant documents, respectively, in the retrieved list. Equation (13) incorporates the effect of positive and negative feedback from relevant and irrelevant documents, respectively, while (14) and (15) are used for normalization. Equation (13) can be explained as follows. As the position vectors are mapped according to the spatial position of the map, each element in $[p_{1x}, p_{2x}, \ldots, p_{c_{\max}x}]$ reflects certain types of nodes. In other word, they represent cluster centers of SOM units. Frequent presence of a nonzero position vector among the relevant documents implies a need to incorporate a higher weight for that particular position vector, and *vice versa*. In this study, root node features are not used for weight-based relevance feedback, because they do not provide promising results with the relevance feedback. The relevance feedback is also enhanced by using query modification [32]. The modified query data $\mathbf{X}_{\text{new}} = \{\mathbf{X}^f_{\text{new}}\mathbf{X}^p_{\text{new}}\}$ is obtained by averaging all the root node inputs $\mathbf{X} = \{\mathbf{X}^f\mathbf{X}^p\} = \{f_1, f_2, \ldots f_k, \mathbf{p}_1, \mathbf{p}_2, \ldots \mathbf{p}_{c_{\max}}\}$ of the query and relevant documents

$$\mathbf{X}^f_{\text{new}} = \frac{1}{R+1}\left(\mathbf{X}^f_q + \sum_{r=1}^{R}\mathbf{X}^f_r\right) \tag{16}$$

$$\mathbf{X}^p_{\text{new}} = \left[\frac{\left(\mathbf{p}^q_1 + \sum_{r=1}^{R}\mathbf{p}^r_1\right)}{\mathbb{S}(\mathbf{p}^q_1) + \sum_{r=1}^{R}\mathbb{S}(\mathbf{p}^r_1)}, \frac{\left(\mathbf{p}^q_2 + \sum_{r=1}^{R}\mathbf{p}^r_2\right)}{\mathbb{S}(\mathbf{p}^q_2) + \sum_{r=1}^{R}\mathbb{S}(\mathbf{p}^r_2)}, \ldots, \frac{\left(\mathbf{p}^q_{c_{\max}} + \sum_{r=1}^{R}\mathbf{p}^r_{c_{\max}}\right)}{\mathbb{S}(\mathbf{p}^q_{c_{\max}}) + \sum_{r=1}^{R}\mathbb{S}\left(\mathbf{p}^r_{c_{\max}}\right)}\right] \tag{17}$$

where $\mathbb{S}$ is the previously defined binary function in (12), R is the number of relevant documents, $\mathbf{X}_r$ is the root node input of the rth relevant document, $\mathbf{X}_q$ is the root node input for the query document, and $\mathbf{p}^r_1$ is the first position vector of X_r. Using the modified query input vector $\mathbf{X}_{\text{new}}$, DR is performed by the following procedures.

Procedure of Document Retrieval Using RF

1) Match the modified query input vector $\mathbf{X}_{\text{new}}$ with the neurons of the top SOM layer using new v_j in (12).
2) Sort the neurons of the top layer in descending order according to similarity with $\mathbf{X}_{\text{new}}$.
3) Go through the sorted neurons in descending order and append their associated documents into the retrieval list until at least N documents are found. N is the number of documents to be retrieved as defined by users.
4) Sort the documents (by matching their root node input with $\mathbf{X}_{\text{new}}$). Return the first N documents to users.

It is worth noting that the above described relevance feedback procedures can be completed in a single or several iterations.

C. Plagiarism Detection Using Retrieval

PD can be carried out in two ways. In the plagiarism detection using retrieval (PDR) method, we use the top layer to retrieve documents in the usual DR operation. This restricts the number of possible document candidates to a much smaller number compared with the total number of documents in the database. We are then able to make local comparison using a paragraph-to-paragraph similarity to detect plagiarism. This method relies on DR in which the "source document" shares a contextual topic similar to plagiarized document. The "source document" can then be confined in the retrieval list. The PDR method is summarized as follows.

1) Extract the tree-structured data of the query document.
2) Retrieve the most relevant documents through normal DR process. User can assign a low value of C in (12) for this task to emphasize local similarity.
3) Compare the query to the retrieved documents using third-level nodes. Comparison of two documents using the third-level nodes can be performed in several ways. The plagiarism distance between the query document D_q and the candidate document D is defined as

$$P_D(D_q, D) = P_{d_i \in D_q}\left(\min_{d_j \in D}(|d_i - d_j|)\right) \tag{18}$$

where (d_i, d_j) are paragraph features (from the third-level nodes) from documents (D_q, D). In (18), the $\min()$ function detects the best match of a query paragraph d_i. The detection results obtained from all paragraphs of the query document are then summed up by the P function. We use the following three P functions in our experiments:

$$\text{min}: \quad P_{\min}(z) = \begin{cases} \min\limits_{i=1\ldots N_q}(z_i), & \text{if } \sum_{i=1}^{N_q}(z_i) \geq 1 \\ \kappa, & \text{otherwise} \end{cases} \tag{19}$$

$$\text{mean}: \quad P_{\text{avg}}(z) = \begin{cases} \dfrac{\sum_{i=1}^{N_q} z_i \times (z_i)}{\sum_{i=1}^{N_q}(z_i)}, & \text{if } \sum_{i=1}^{N_q}(z_i) \geq 1 \\ \kappa, & \text{otherwise} \end{cases} \tag{20}$$

$$\text{min} + \frac{1}{2}\,\text{mean}: \quad P_{\text{hyb}}(z) = P_{\min}(z) + \frac{1}{2}P_{\text{avg}}(z) \tag{21}$$

where

$$\mathbb{Z}(z_i) = \begin{cases} 1, & \text{if } z_i < \varepsilon \\ 0, & \text{otherwise} \end{cases}$$

and N_q is the number of paragraphs in query document D_q. Here, ε is an offset value for detecting plagiarism, and κ is a predefined large value for P function when no match is found.
4) Find the candidate documents having a P value lower than κ. Sort them according to ascending value of P and return them to the user.

Besides the above detection process, a simple retrieval can be performed in case that we do not want to filter results by setting offset value of ε. To perform a retrieval by using local sorting, simpler functions $P_{\min}(z) = \min\limits_{i=1\ldots N_q}(z_i)$, $P_{\text{avg}}(z) = \overline{z_i}$, and $P_{\text{hyb}}(z) = P_{\min}(z) + (1/2)P_{\text{avg}}(z)$ are used.

D. Plagiarism Detection Using Local Association

In the plagiarism detection using local association (PDLA) approach, the bottom layer is used for directly retrieving the suspected documents without taking global similarity between the two documents into account. Paragraphs of all documents in the database are clustered in the bottom layer. In the PDLA method, we compare the paragraphs of the query document with the neurons to retrieve the possible candidates of "source documents" of plagiarism. The procedures are summarized as follows.

1) Extract the tree-structured data of the query document. Say it has L number of nodes in the third level.
2) For each of the third-level nodes, find the best matching neuron. Stack index of associated document data together with index of third-level nodes in the tree. Thus, a separate list is made for each of the L nodes. In the case that no associated document is found in the neuron, the node is ignored.
3) Using the list of each node, compute the $L1$ distance between the third-level node of the query and that from the associated documents

$$Q_d = |d_q - d|\,. \tag{22}$$

4) Make a new list by merging the document indexes from L number of lists generated from the third-level nodes of query, and their corresponding Q_d value. In this combined list, duplicate presence of an index is possible because of multiple matches among the third-level nodes from the query and an associated document.
5) Remove the index of a document having a Q_d value above an offset τ.
6) Find the duplicate indexes, and merge them by computing a new value of Q_d as

$$Q_d^* = \frac{\overline{Q_d}}{\sqrt{N_q}} \tag{23}$$

where N_q is the number of duplicates found. In (23), the plagiarism distance Q_d is reduced, because multiple matches are found between the query and an associated document.
7) Sort the list according to ascending value of Q_d and return the associated documents.

These two detection methods are implemented in our study for thorough comparisons. The choice for the value of ε and τ depends on the feature value of third-level nodes. We will discuss both ε and τ in detail in the next section.

V. Experimental Results

The document database, named "*Html_CityU1*," consists of 26 categories. Each category contains 400 documents making a total of 10 400 documents. There are 1040 test documents randomly selected from the 26 categories, i.e., 26 × 40. The remaining 9360 documents were used for training. In order to provide a real-life and demanding testing platform, we have established this database with document size ranged from one page to about 200 pages, and few hundred words to over 20 000 words. For each of the 26 categories, 400 documents are retrieved from Google using a set of keywords. The set of keywords for a category is different from that of the others, but some of the keywords may be shared among different categories.[1] After the MLSOM was trained, the test set was used to verify the retrieval

[1]The database can be downloaded from www.ee.cityu.edu.hk/~twschow/Html_CityU1.rar

TABLE I
Distribution of Nodes in Data Sets

Level	1	2	3	All levels
Max. number of children nodes	277	102	0	102
Total number of nodes in training set	9360	19099	114223	142682
Total number of nodes in test set	1040	1954	11368	14362

accuracy. Node distribution in different tree levels of the document data is listed in Table I. According to the node distribution, the sizes of the top, middle, and bottom layers of MLSOM were set at 30 × 30, 36 × 36, and 42 × 42. The initial learning rate was set to 0.3. The initial radius of the neighborhood function was set to half-length of the square grid at a SOM layer. The number of total training iterations was set to 28 080, 28 459, and 114 223 (which are the rounded multiples of the number of data nodes in the corresponding level) for the top, middle, and bottom layers, respectively. In this study, the values of the above parameters were observed to be a good choice. All simulations were performed using MATLAB 7 on a PC with Intel 1.8-GHz and 2-GB memory.

A. Document Retrieval

In this section, we present detailed experimental results from the MLSOM-based retrieval system. We will also compare the results with direct method using tree data. In the direct method of retrieval, the query tree data D_q is compared in the database, and documents with lower tree distance are returned to the user. Two document tree data (D_q, D) are compared with the following tree distance function T_d:

$$T_d\left(D_q, D\right) = T_d\left(n_1^q, n_1\right) \tag{24}$$

$$T_d\left(n_x^q, n_y\right) = C \times d_f\left(F_{n_x}^q, F_{n_x}\right) + \frac{1}{\delta(n_x^q)} \sum_{i \in \beta(n_x^q)} \min_{j \in \beta(n_x)} T_d\left(n_i^{qx}, n_j^y\right) \tag{25}$$

where n_1^q is the root node of the query tree, C is the weight parameter similar to (12), d_f is the distance function, $F_{n_x}^q$ is the feature vector for the xth node of the query, $\delta(n_x^q)$ is the number of child nodes of the node n_x^q, n_i^{qx} is a child node of n_x^q, and $\beta(n_x^q)$ contains the child nodes' index of the node n_x^q. The distance function T_d is a recursive one; the first part contains distance in global node's feature, and the second part contains the average distance in local child nodes through recursion. C is the same weight as used in (12). We used $L1$, $L2$, and cosine distance function for d_f as follows:

$$L1: \quad d_f\left(a, b\right) = \sum_i |a_i - b_i|. \tag{26}$$

$$L2: \quad d_f\left(a, b\right) = \sqrt{\sum_i |a_i - b_i|^2}. \tag{27}$$

$$\text{cosine}: \quad d_f\left(a, b\right) = -\sum_i a_i \cdot b_i. \tag{28}$$

In (12), we used $L1$ distance (in the first part) to compute the similarity between the nodes. We conducted simulations using $L2$ and cosine distance. We also include results from the LSI approach [15] without using SOM. In the LSI approach, two documents are compared directly by calculating the distance between their LSI features. We do not include the classical VSM model in our comparative study, as it is too computational and resource demanding for implementation. To summarize the retrieval results, we used each document data from the test set as a query. The retrieval accuracy is computed by precision defined as follows:

$$\text{precision} = \frac{\text{number of correct documents retrieved}}{\text{number of total documents retrieved}}. \qquad (29)$$

Fig. 4 summarizes the precision results of retrieving 360 documents. It also compares the effect of different values of parameter C in (12). Fig. 4 includes the results for using two different normalization methods for document feature: "mean norm" and "VSM norm." For each normalization, Fig. 4 shows the results of the first retrieval as well as the retrieval result after relevance feedback. The relevance feedback can consistently improve the retrieval results using "mean norm." In our experiments, the "max norm" did not perform well compared with other two and its results are excluded here. With the increase in the number of documents retrieved, the precision drops with different slopes depending on the approach/parameter settings. The complete precision plots for retrieving unto 360 documents are shown in Fig. 4.

It is noted that the above results were obtained by using $L1$ distances in all cases. Fig. 5 summarizes their precisions for using different combinations of normalization and distance functions. The distance function $L1$ and $L2$ deliver similar performance, and $L2$-based results are not included separately. The choice of normalization and distance function is not straightforward. For example, in using direct method and "VSM norm," the cosine distance function performs better when a larger number of retrieving documents is considered, and $L1$ distance function performs better for retrieving a small number of documents. On the other hand, when mean normalization is considered for the direct method, the scenario changes. Again, the whole scenario changes when we consider a SOM method instead of a direct method.

It is observed that the SOM approach using "mean norm" delivers better results than the direct method. In this case, the superior performance exhibited by the SOM is believed to be attributed to the coexistence of the SOM properties of "organizing/clustering," "topological ordering," and "nonlinear projection." It is also worth noting that the normalization of a feature plays an important rule in making choice between the SOM and direct methods. While the SOM approach performs better with "mean norm," the direct method usually performs better with "VSM norm." Analyzing the above results, it is obvious that the choices of an appropriate value of C and normalization method are dependent upon the type of the retrieval method. Overall, the SOM approach performs the best using "mean norm," "$C = 1$," and $L1$ distance. Using these settings, Table II summarizes the precision results of different approaches together with their computational time. The table also summarizes the results of "recall" that is defined as follows:

$$\text{recall} = \frac{\text{number of correct documents retrieved}}{\text{number of total documents in relevant class}}. \qquad (30)$$

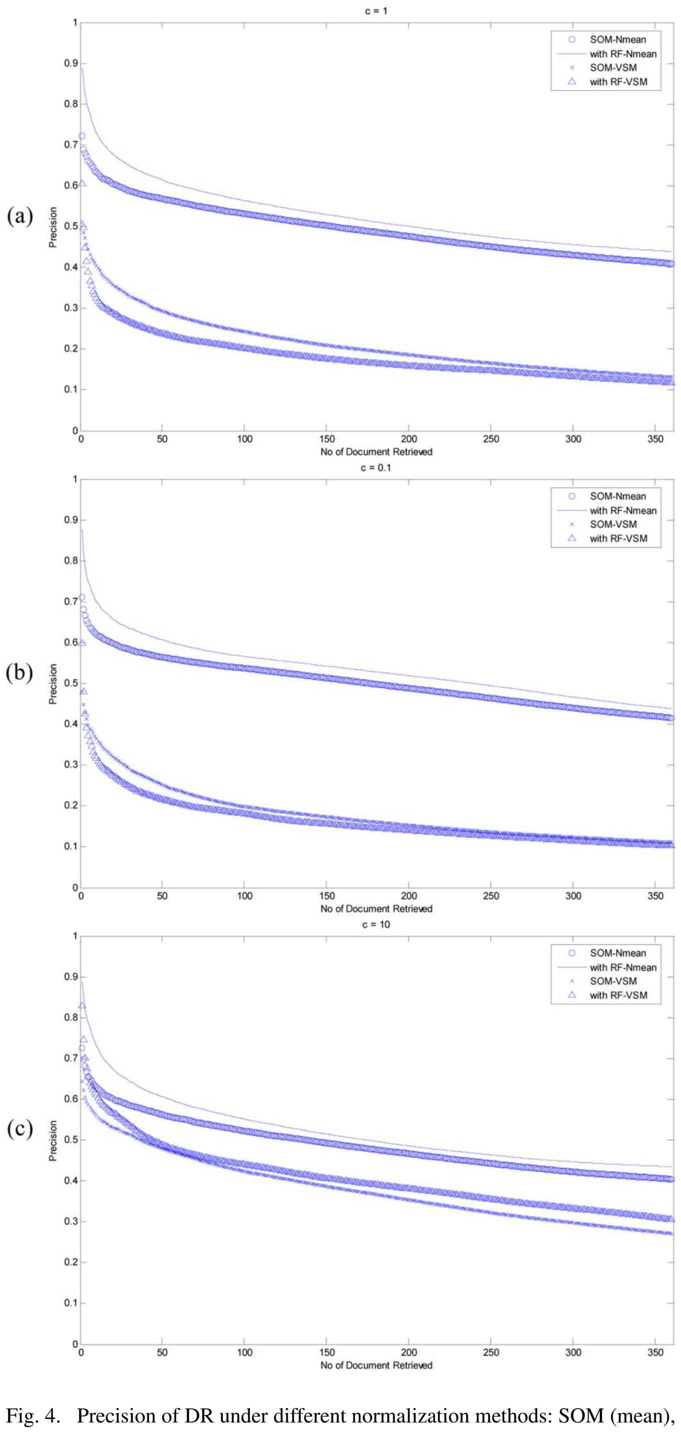


Fig. 4. Precision of DR under different normalization methods: SOM (mean), SOM with RF (mean), SOM (VSM), SOM with RF (VSM) for different c. (a) $c = 1$, (b) $c = 0.1$, and (c) $c = 10$.

The reported results using precision and recall indicate that the proposed SOM approach with tree-structure features can provide better results compared with traditional LSI approach that uses flat features. It is noted that the basic feature of our approach and traditional approach lies in the same root of the term-frequency information. Our approach incorporates the same feature in a global-to-local structural way. As an advantage of this approach, users can easily specify the relative emphasis between global and local characteristics by choosing

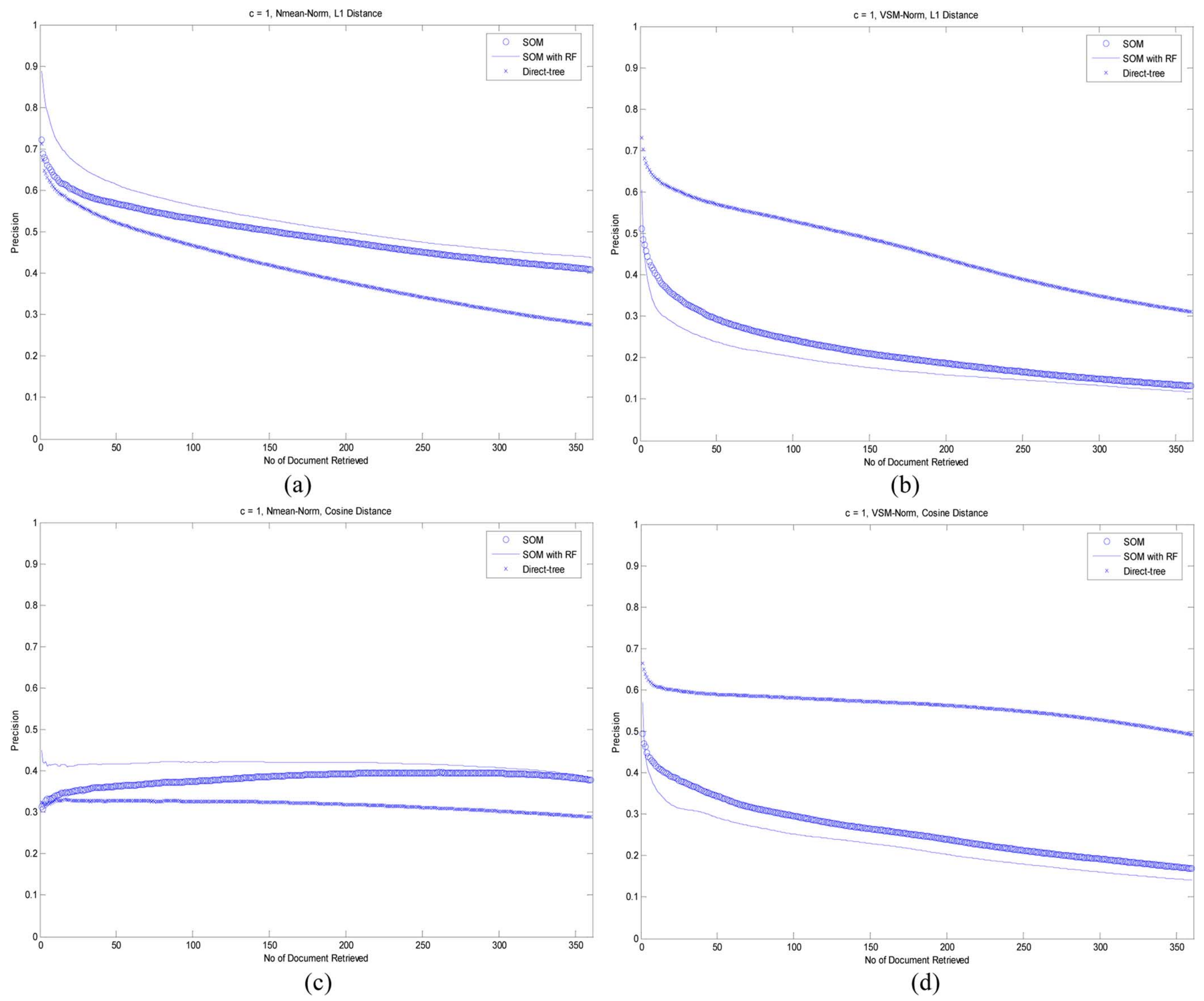


Fig. 5. Precision of document retrieval by SOM, SOM with RF and direct method for different distance functions and normalization methods. (a) Mean norm, $L1$ distance. (b) VSM norm, $L1$ distance. (c) Nmean norm, cosine distance. (d) VSM norm, cosine distance. $C = 1$ for all cases.

TABLE II
COMPARATIVE RESULTS ON DIFFERENT APPROACHES FOR DOCUMENT RETRIEVAL

Feature	Query Type	Query Time (sec)	No of retrieved Documents					
			10	40	360	10	40	360
			Precision			Recall		
Tree-structured feature	MLSOM	0.39	63.29	57.60	40.87	1.75	6.40	40.87
	RF in MLSOM	0.40	72.87	62.95	43.79	2.02	6.99	43.79
	Direct Method	49.23	63.44	58.16	31.11	1.76	6.46	31.11
Flat-feature	LSI approach	0.48	62.13	55.85	29.58	1.73	6.21	29.58

an appropriate value of C during retrieval. These advantages are discussed in Sections V-B and V-C.

Table III summarizes the retrieval results from MLSOM for using different types of features, i.e., global, local, and tree structured. For global features, the PCA features of the root nodes are used with the nodes at levels 2 and 3 being ignored. For local feature, we exclude the features of the root nodes. It is evident that the tree-structured data perform better compared with global or local features individually. To ensure the generalization of the proposed system, we also included the retrieval results from the tenfold cross-validation sets. Each of the validation sets consists of 1040 documents without any overlap among the sets, so that all data are used for testing. Table IV summarizes the precision and the recall results from the tenfold cross-validations sets. The results from ten validation sets are similar indicating a stable performance of the system. At last, we study the robustness of the MLSOM over SOM initialization. We summarize the precision results over different training sessions in Table V. It is observed that the MLSOM is able to deliver similar results under different initializations.

TABLE III
RETRIEVAL RESULTS FROM MLSOM USING DIFFERENT TYPES OF FEATURES

Feature type	Query Type	No of retrieved Documents					
		10	40	360	10	40	360
		Precision			Recall		
Global	MLSOM	58.87	53.93	37.80	1.64	5.99	37.80
	RF	66.49	57.22	39.37	1.85	6.36	39.37
Local	MLSOM	47.62	43.82	21.78	1.32	4.87	21.78
	RF	51.71	44.56	20.57	1.44	4.95	20.57
Tree-Structured	MLSOM	63.29	57.60	40.87	1.75	6.40	40.87
	RF	72.87	62.95	43.79	2.02	6.99	43.79

TABLE IV
RETRIEVAL RESULTS (PRECISION) FROM MLSOM USING TENFOLD CROSS-VALIDATION SETS

	Cross-validation set	No of retrieved Documents						
		10	40	360		10	40	360
MLSOM	1	63.47	57.62	40.52	6	61.57	56.03	39.95
RF		73.53	63.01	43.45		72.78	62.39	42.88
MLSOM	2	62.55	57.73	40.87	7	65.53	60.16	41.16
RF		72.93	63.29	43.90		73.88	64.55	43.73
MLSOM	3	63.88	58.78	41.13	8	62.70	57.21	40.35
RF		73.32	63.71	43.71		71.74	61.70	43.05
MLSOM	4	62.28	57.45	40.14	9	61.49	56.62	39.86
RF		72.01	62.38	42.55		72.54	62.57	42.79
MLSOM	5	61.92	56.92	39.90	10	63.29	57.60	40.87
RF		71.48	61.60	42.57		72.87	62.95	43.79
MLSOM	**Average**	**62.87**	**57.61**	**40.48**				
RF		**72.71**	**62.81**	**43.24**				

TABLE V
PRECISION RESULTS FROM DIFFERENT TRAINING SESSIONS

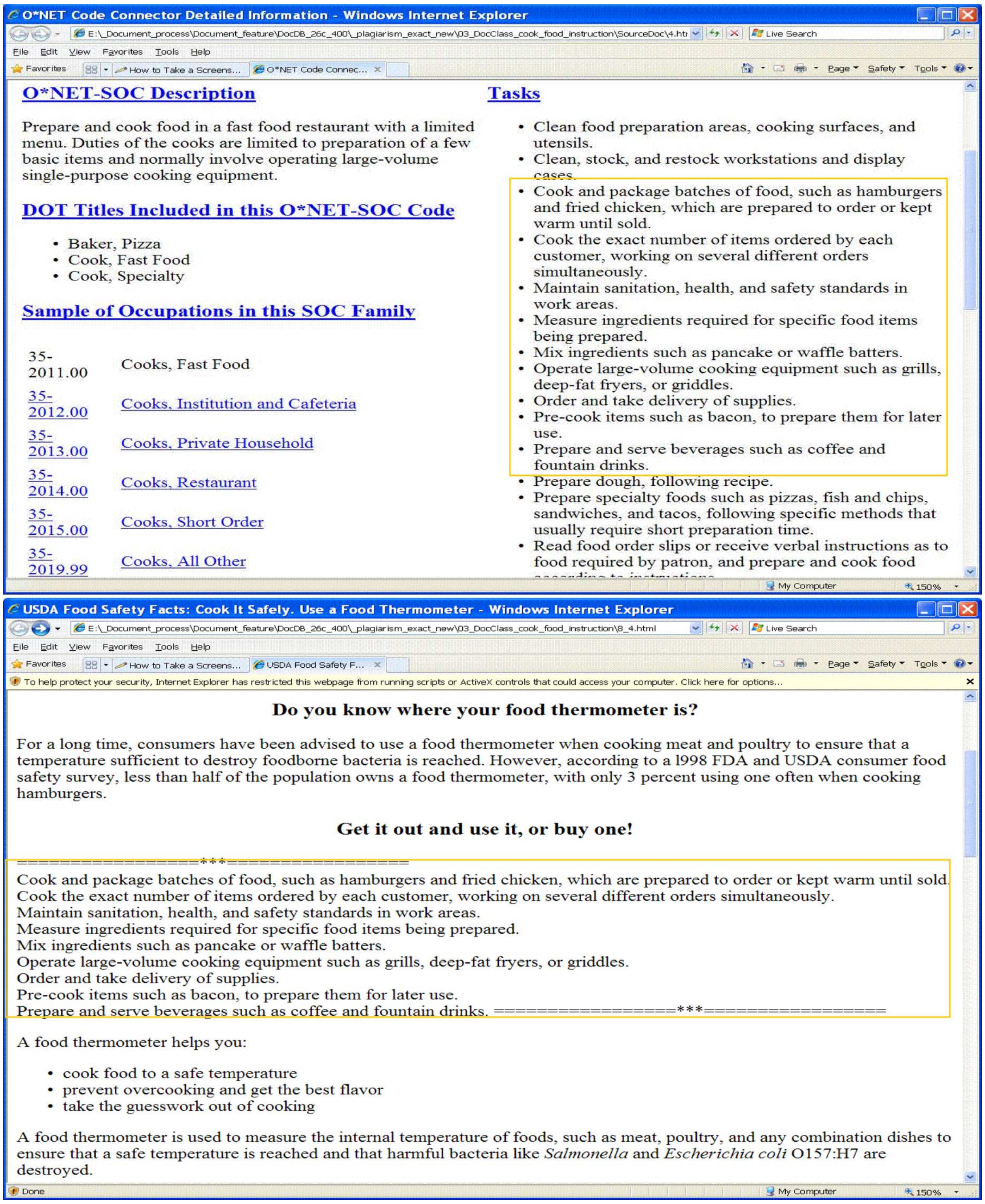
Training Session	Query Type	No of retrieved Documents		
		10	40	360
1	MLSOM	63	58	41
	RF	73	63	44
2	MLSOM	63	58	41
	RF	73	63	44
3	MLSOM	63	57	40
	RF	72	62	43
4	MLSOM	63	58	41
	RF	73	63	44

The computational time listed in Table II excludes the feature extraction process for query that was conducted offline. It is obvious that using tree-structured features, the direct method is computationally demanding, because of the recursive many-to-many matching in the second part of (25). This procedure is computationally heavier compared to the one-to-one matching in (12) by MLSOM. Table II indicates that the required computational time of MLSOM can be up to 120 times shorter than the time of the direct method. Thus, the SOM algorithm is capable of reducing the massive computational cost, and is able to maintain similar or better retrieval accuracy.

B. Plagiarism Detection Using Retrieval

1) Rank of Source Document: In DR, the comparative performance between tree-structured feature and traditional LSI may not be good enough, because document classes are collected using Google, which is based on keyword-based searching that reflects only global similarity among documents. The advantage of the proposed method can be easily comprehended when we test some DR using plagiarized documents. In making the plagiarized document sets, only a small part (approximately equal to a paragraph) of a test document was copied from a document of the training set called *source document*. Thus, a plagiarism document set is made from our test set, consisting of 78 documents, three documents from each category. Fig. 6 shows an example of such copying. We also made a second set from the above plagiarism set by changing sentences in the copied text. These minor changes including change of tense, active to passive, few words, etc., show an example of such copying. Unless specified, the following experiments were conducted on the first plagiarism set.

In this section, 500 documents were retrieved using each of the plagiarized documents as a query, and we observe the rank of the *source document* in the results. The rank indicates the position of *source document* in the retrieved document list. A lower rank value is desired for easy detection. Table VI shows the average ranks of a *source document* in the retrieval results for 78


O*NET Code Connector Detailed Information - Windows Internet Explorer
E:_Document_process\Document_feature\DocDB_26c_400_plagiarism_exact_new\03_DocClass_cook_food_instruction\SourceDoc\4.htm
File Edit View Favorites Tools Help
Favorites
How to Take a Screens...
O*NET Code Connec...
Page Safety Tools
My Computer
150%
USDA Food Safety Facts: Cook It Safely. Use a Food Thermometer - Windows Internet Explorer
E:_Document_process\Document_feature\DocDB_26c_400_plagiarism_exact_new\03_DocClass_cook_food_instruction\8_4.html
USDA Food Safety F...
To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer. Click here for options...
Done

O*NET-SOC Description

Prepare and cook food in a fast food restaurant with a limited menu. Duties of the cooks are limited to preparation of a few basic items and normally involve operating large-volume single-purpose cooking equipment.

DOT Titles Included in this O*NET-SOC Code

- Baker, Pizza
- Cook, Fast Food
- Cook, Specialty

Sample of Occupations in this SOC Family

35-2011.00	Cooks, Fast Food
35-2012.00	Cooks, Institution and Cafeteria
35-2013.00	Cooks, Private Household
35-2014.00	Cooks, Restaurant
35-2015.00	Cooks, Short Order
35-2019.99	Cooks, All Other

Tasks

- Clean food preparation areas, cooking surfaces, and utensils.
- Clean, stock, and restock workstations and display cases.
- Cook and package batches of food, such as hamburgers and fried chicken, which are prepared to order or kept warm until sold.
- Cook the exact number of items ordered by each customer, working on several different orders simultaneously.
- Maintain sanitation, health, and safety standards in work areas.
- Measure ingredients required for specific food items being prepared.
- Mix ingredients such as pancake or waffle batters.
- Operate large-volume cooking equipment such as grills, deep-fat fryers, or griddles.
- Order and take delivery of supplies.
- Pre-cook items such as bacon, to prepare them for later use.
- Prepare and serve beverages such as coffee and fountain drinks.
- Prepare dough, following recipe.
- Prepare specialty foods such as pizzas, fish and chips, sandwiches, and tacos, following specific methods that usually require short preparation time.
- Read food order slips or receive verbal instructions as to food required by patron, and prepare and cook food

Do you know where your food thermometer is?

For a long time, consumers have been advised to use a food thermometer when cooking meat and poultry to ensure that a temperature sufficient to destroy foodborne bacteria is reached. However, according to a 1998 FDA and USDA consumer food safety survey, less than half of the population owns a food thermometer, with only 3 percent using one often when cooking hamburgers.

Get it out and use it, or buy one!

==================***==================
Cook and package batches of food, such as hamburgers and fried chicken, which are prepared to order or kept warm until sold.
Cook the exact number of items ordered by each customer, working on several different orders simultaneously.
Maintain sanitation, health, and safety standards in work areas.
Measure ingredients required for specific food items being prepared.
Mix ingredients such as pancake or waffle batters.
Operate large-volume cooking equipment such as grills, deep-fat fryers, or griddles.
Order and take delivery of supplies.
Pre-cook items such as bacon, to prepare them for later use.
Prepare and serve beverages such as coffee and fountain drinks. ==================***==================

A food thermometer helps you:

- cook food to a safe temperature
- prevent overcooking and get the best flavor
- take the guesswork out of cooking

A food thermometer is used to measure the internal temperature of foods, such as meat, poultry, and any combination dishes to ensure that a safe temperature is reached and that harmful bacteria like *Salmonella* and *Escherichia coli* O157:H7 are destroyed.

Fig. 6. Typical example of plagiarized documents (Top) Source document. (Bottom) Plagiarized document. The highlighted parts are the plagiarized text.

queries of plagiarism documents. The results summarize the effect of emphasis between global and local (different values of C) in detecting plagiarized document. Table VI includes *failed detection* indicating the percentage of retrieval cases that failed to include the *source document*. Based on the *average rank* and *failed detection*, a *composite rank* is calculated. It is clear that we can achieve better rank of *source document* by equally emphasizing $(C = 1)$ on global and local characteristics of a document. Emphasizing more on global characteristics causes higher *failed detection*. At last, we investigate the effect of presence of *source document* in the first number of documents from the retrieved list. Table VII summarizes the results of presence of *source document* when it is 1, 20, and 50. In this experiment, we used normal retrieval method without incorporating any special technique for PD. In the next section, we investigate the positive effect of incorporating a local sorting procedure.

2) Retrieval With Local Sorting: To effectively retrieve the *source document* of plagiarism, we incorporate an additional sorting process after the retrieval. As described in Section IV-C, the retrieved documents are sorted according to their similari-

TABLE VI
RANKS OF SOURCE DOCUMENT OF PLAGIARISM IN NORMAL DOCUMENT RETRIEVAL RESULTS

	Rank statistics					
	Average Rank	Standard Deviation	Maximum	Minimum	*Failed Detection* (%)	Composite Rank (Avg*Failed)
C=10	110.43	117.32	451	1	39.74	43.88
C=1	108.58	114.07	465	1	35.90	38.97
C=0.1	130.42	123.83	422	1	35.89	46.81
After paragraph sorting						
C=10	9.85	25.76	114	1	39.74	3.92
C=1	8.72	23.53	107	1	35.90	3.13
C=0.1	11.36	27.55	127	1	35.89	4.08

TABLE VII
PRESENCE OF SOURCE DOCUMENT OF PLAGIARISM IN RETRIEVAL

	Number of documents investigated		
	1	20	50
C=10	4%	15%	26%
C=1	3%	18%	24%
C=0.1	3%	15%	22%
After paragraph sorting			
C=10	42%	54%	54%
C=1	46%	58%	59%
C=0.1	46%	55%	56%

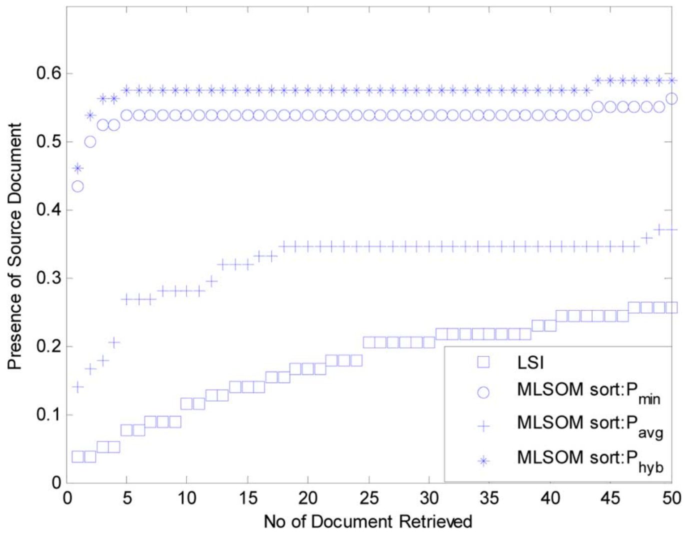
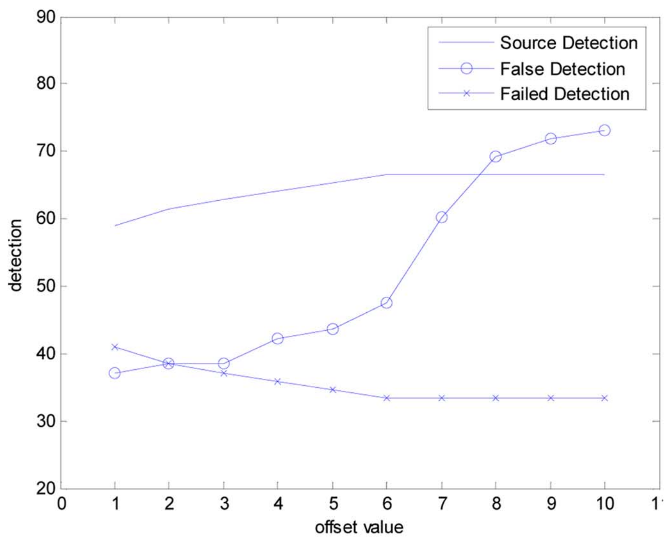


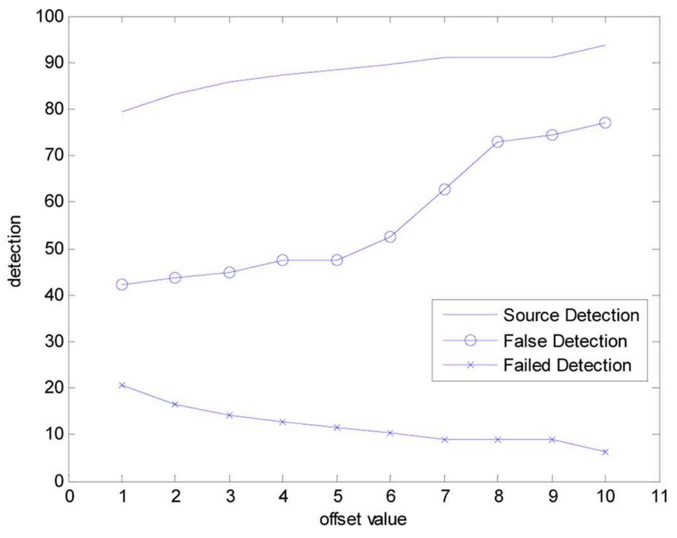
Fig. 7. Presence of *source document* of plagiarism in retrieval against the number of document investigated.

ties in terms of the third-level nodes. Instead of using the functions in Section IV-C, we used a modified and simple set of sorting functions: $P_{\min}(z) = \min_{i=1\ldots N_q}(z_i)$, $P_{\text{avg}}(z) = \overline{z_i}$, and $P_{\text{hyb}}(z) = P_{\min}(z) + (1/2)P_{\text{avg}}(z)$. Using the local sorting P_{hyb}, Table VI shows new rank results. Table VII includes new results of presence of *source document* when N_I is considered to be 1, 20, and 50. The results are summarized for different "C" values in (12). In our results, mean norm and "$C = 1$" are the best retrieval setting for PD. Using these settings and the local sorting, Fig. 7 shows the complete plots of presence of *source document* for different sorting functions in MLSOM retrieval. While $P_{\min}$ function fits best for detecting a single paragraph, P_{avg} function is more effective in detecting multiple plagiarized sections. The function P_{hyb}, being hybrid of the above two, in general works well. Fig. 7 also includes a result from the plain LSI approach without incorporating any sorting. These results demonstrate how tree-structured features possess the advantage over the traditional global flat features. Thus, the tree-structured features serve us better by including local characteristics of a document as well as users' flexibility of assigning emphasis between global and local characteristics of a document.

3) Automatic Plagiarism Detection: In this subsection, we consider PD in an automatic way rather than looking for probable *source document* in the retrieval results. To make the binary decision on the presence of *source document*, we follow the procedures in Section IV-C, and set a small offset value for ε that is used for local sorting process. Here, 1000 documents are retrieved and sorted by the local sorting procedures. Table VIII shows the results of automatic PD against different offset values of ε. The table also includes *false detection*, which indicates the case when other than *source document* is detected as plagiarism source. It is observed that a higher value of ε results in unexpected *false detection*. We observed that the standard deviation of average feature value $(\overline{f_i})$ over all third-level nodes $(\text{std}\,(\overline{f_i})$ is 3.9. Using a value of ε around 3.9 delivers the best result that maintains a good balance between *source detection* and *false detection*. Fig. 8(a) shows the plot of *source detection*, *false detection*, and *failed detection* against different values of ε. With an increase of ε, the *source/false detection* increases, while the *failed detection* drops. An optimum choice of detection is to find a value of ε giving *source detection* as high as possible, and keeping a *false/failed detection* as low as possible. Fig. 8 shows that a value of ε around $\text{std}\,(\overline{f_i})$ exhibits such a balanced performance. Table VIII also includes the detection results from the second set of plagiarism documents. Using $\varepsilon = 5$, the *source detection* rate drops by 3.85%, which is due to the modification on the copied text.



(a)



(b)

Fig. 8. PD statistics against different offset value: (a) plagiarism by retrieval and (b) plagiarism through local association.

TABLE VIII
DETECTION RESULTS OF PDR/PDLA AGAINST DIFFERENT VALUES OF ε OR τ

ε or τ	PDR Auto Plagiarism detection (in %)			PDLA Auto Plagiarism detection (in %)		
	Source Detection	*False Detection*	*Failed Detection*	*Source Detection*	*False Detection*	*Failed Detection*
20	69.23	98.71	30.77	93.58	100	6.41
10	66.66	73.07	33.33	93.58	76.92	6.41
5	65.38	43.58	34.61	88.46	47.43	11.53
1	58.97	37.17	41.02	79.48	42.3	20.51
0.1	55.12	35.89	44.87	75.64	41.02	24.35
0.001	55.12	35.89	44.87	75.64	41.02	24.35
2nd of plagiarism documents						
20	69.23	98.71	30.76	92.3	100	7.692
10	65.38	73.07	34.61	89.74	76.92	10.25
5	61.53	43.58	38.46	85.89	46.15	14.1
1	41.02	35.89	58.97	52.56	39.74	47.43
0.1	19.23	32.05	80.76	28.2	35.89	71.79
0.001	19.23	32.05	80.76	28.2	35.89	71.79

TABLE IX
COMPUTATIONAL TIME (IN SECOND) FOR PLAGIARISM DETECTION

	Average	Std
By PDLA	0.6317	0.2855
By PDR (1000)	7.4531	3.4633
By PDR (500)	4.0251	3.2811

C. *Plagiarism Detection Through Local Association*

This section investigates a more effective PD method using the bottom SOM layer of MLSOM. This approach, described in Section IV-D, does not need to go through normal DR process. The bottom layer is used to organize the third-level nodes representing local paragraphs. For a given query document, the third-level nodes are matched with the neurons of the bottom layer. The associated documents only from the winner neurons are returned with a rank according to the distance between paragraphs. Table VIII shows the results obtained from different values of offset τ. The results show that an 88% *source detection* and a *false detection* below 48% are maintained. But the *false detection* is slightly higher than that of the previous approach. The basic difference between the two approaches is that the first approach uses the global document similarity to restrict a list of suspected documents, while the second approach uses local similarities to search through the clusters of all documents at the bottom layer. As a result, the *source detection* rate is increased by 23%. Similarly to the previous approach, we found a good tradeoff between *source detection* and *false detection* by selecting the value of τ around $\mathrm{std}\left(\overline{f_i}\right)$. Table VIII presents the results of the second set of plagiarism documents that include modified copied text. Using $\tau = 5$, the *source detection* rate is 2.57% less than that of the original plagiarism data set. The performance difference between the two plagiarism sets is slightly less than that from using the PDR method. Compared with PDR with PDLA, the above observations indicate that PDLA is a more effective method to detect modified type of plagiarism text.

The comparative computational time is shown in Table IX. The listed time is the searching time that excludes the feature extraction of a query document. The average computational time of segmentation and feature extraction for a query document is around 1 s making real-time application possible. The required computational time of PD varies from document to document depending upon the size of a document. In summary, PDLA can be up to ten times faster than PDR, because local association does not need to compare a paragraph of a query to all paragraphs in the database. On the other hand, even though candidate documents are limited, the PDR needs to handle a many-to-many matching by (18), which is rather computationally demanding. However, in handling a massive increase in the database size, the computational time of PDR will not significantly increase, because it restricts its plagiarism searching to a limited retrieved document set. On the other hand, PDLA may need an increase in time, as the number of locally associated documents will increase accordingly with the size of a database. However, the PDLA method is more effective in detecting plagiarism.

For a very large scale implementation of DR and PD, it is difficult to process all documents in a single SOM module, or to update SOM when the database is required to be updated after a period of time. We can consider modular SOM approach, in which database of different domains can be processed over many MLSOM modules. This can be a future work on DR using SOM. In our future work, we will investigate detecting sophisticated plagiarism that involves more than simple copy of words in a paragraph.

VI. Conclusion

A new approach of DR and PD using tree-structured document representation and MLSOM is proposed. We have shown that tree-structured representation enhances the retrieval accuracy by incorporating local characteristics with traditional global characteristics. The proposed method provides flexibility to assign relative emphasis between the local and global characteristics of a document. Together with the support of relevance feedback, the proposed system provides improved accuracy in DR. Hierarchical organization of global and local characteristics enables the MLSOM to be used for PD. Two methods of PD are proposed. The first one is an extension of the DR method with additional local sorting. The second method uses document association on the bottom layer of MLSOM. Our obtained results indicate that the second method consistently delivers higher detection rate.

Computational cost is an important issue, because this application usually deals with a large database. In our work, the MLSOM serves as an efficient computational solution for practical implementation; its computational time can be up to 120 times faster than the direct method.

Acknowledgment

The authors would like to thank the anonymous reviewers for providing useful comments.

References

[1] S. Deng and H. Peng, "Document classification based on support vector machine using a concept vector model," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell.*, Dec. 2006, pp. 473–476.

[2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Disc.*, vol. 2, no. 2, pp. 121–167, 1998.

[3] T. Onoda, H. Murata, and S. Yamada, "SVM-based interactive document retrieval with active learning," *New Gen. Comput.*, vol. 26, no. 1, pp. 49–61, 2008.

[4] R. Nallapati, "Discriminative models for information retrieval," in *Proc. 27th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2004, pp. 64–71.

[5] R. T. Freeman and H. Yin, "Web content management by self-organization," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1256–1268, Sep. 2005.

[6] K. Monostori, A. Zaslavsky, and H. Schmidt, "MatchDetectReveal: Finding overlapping and similar digital documents," in *Proc. Inf. Resources Manage. Assoc. Int. Conf. Challenges Inf. Technol. Manage. 21st Century*, Anchorage, AK, 2000, pp. 955–957.

[7] S. B. J. Davis and H. Garcia-Molina, "Copy detection mechanisms for digital documents," in *Proc. ACM SIGMOD Annu. Conf.*, 1995, pp. 398–409.

[8] N. Shivakumar and H. Garcia-Molina, "Building a scalable and accurate copy detection mechanism," in *Proc. 1st ACM Conf. Digit. Libraries*, Bethesda, MD, 1996, pp. 160–168.

[9] A. Si, H. V. Leong, and R. W. H. Lau, "CHECK: A document plagiarism detection system," in *Proc. ACM Symp. Appl. Comput.*, Feb. 1997, pp. 70–77.

[10] R. B. Yates and B. R. Neto, *Modern Information Retrieval*. Reading, MA: Addison-Wesley/Longman, 1999.

[11] J. Zobel and A. Moffat, "Exploring the similarity space," *ACM SIGIR Forum*, vol. 32, no. 1, pp. 18–34, 1998.

[12] C. Buckley and J. Walz, "SMART at TREC-8," in *Proc. Text Retrieval Conf.*, 1999, pp. 577–582.

[13] S. Robertson and S. Walker, "Okapi/Keenbow at TREC-8," in *Proc. Text Retrieval Conf.*, vol. 99, pp. 151–162.

[14] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Melbourne, Australia, 1998, pp. 275–281.

[15] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, pp. 391–407, 1990.

[16] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Rev.*, vol. 37, no. 4, pp. 573–595, 1995.

[17] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," *J. Comput. Syst. Sci.*, vol. 61, no. 2, pp. 217–235, 2000.

[18] T. Honkela, S. Kaski, K. Lagus, and T. Kohonen, "WEBSOM-self-organizing maps of document collections," in *Proc. Workshop Self-Organizing Maps*, Espoo, Finland, Jun. 4–6, 1997, pp. 310–315.

[19] N. Ampazis and S. Perantonis, "LSISOM, a latent semantic indexing approach to self-organizing maps of document collections," *Neural Process. Lett.*, vol. 19, no. 2, pp. 157–173, 2004.

[20] R. Zhao and W. Grosky, "Narrowing the semantic gap—Improved text-based web document retrieval using visual features," *IEEE Trans. Multimedia*, vol. 4, no. 2, pp. 189–200, Jun. 2002.

[21] J. Bear, D. Israel, J. Petit, and D. Martin, "Using information extraction to improve document retrieval," in *Proc. 6th Text Retrieval Conf.*, Gathiersburg (MD), Nov. 1997, pp. 367–377.

[22] R. Gaizauskas and Y. Wilks, "Information extraction: Beyond document retrieval," *J. Documentation*, vol. 54, no. 1, pp. 70–105, 1998.

[23] H. Chen, K. J. Lynch, K. Basu, and D. T. Ng, "Generating, integrating, and activating thesauri for concept-based document retrieval," *IEEE EXPERT*, ser. Artificial Intelligence in Text-Based Information Systems, vol. 8, no. 2, pp. 25–34, Apr. 1993.

[24] N. Rooney, D. Patterson, M. Galushka, and V. Dobrynin, "A scaleable document clustering approach for large document corpora," *Inf. Process. Manage.*, vol. 42, pp. 1163–1175, 2006.

[25] S. M. Chen, Y. J. Horng, and C. H. Lee, "Document retrieval using fuzzy-valued concept networks," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 31, no. 1, pp. 111–118, Feb. 2001.

[26] Y. J. Horng, S. M. Chen, Y. C. Chang, and C. H. Lee, "A new method for fuzzy information retrieval based on fuzzy hierarchical clustering and fuzzy inference techniques," *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 2, pp. 216–228, Apr. 2005.

[27] D. H. C. Du, S. Ghanta, K. J. Maly, and S. M. Sharrock, "An efficient file structure for document retrieval in the automated office environment," *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 2, pp. 258–273, Jun. 1989.

[28] A. Georgakis, C. Kotropoulos, A. Xafopoulos, and I. Pitas, "Marginal median SOM for document organization and retrieval," *Neural Netw.*, vol. 17, no. 3, pp. 365–377, 2004.

[29] D. Merkl, S. H. He, M. Dittenbach, and A. Rauber, "Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization," in *Proc. 4th Workshop Self-Organizing Maps/Adv. Self-Organizing Maps*, Kitakyushu, Japan, Sep. 2003, pp. 293–298.

[30] Z. Wang, M. Hagenbuchner, A. C. Tsoi, S. Y. Cho, and Z. Chi, "Image classification with structured self-organization map," in *Proc. Int. Joint Conf. Neural Netw.*, 2002, vol. 2, pp. 1918–1923.

[31] M. Hagenbuchner, "Extension and evaluation of adaptive processing of structured information using artificial neural networks," Ph.D. dissertation, Faculty Inf., Univ. Wollongong, Wollongong, N.S.W., Australia, 2002.

[32] T. W. S. Chow, M. K. M. Rahman, and S. Wu, "Content based image retrieval by using tree-structured features and multi-layer SOM," *Pattern Anal. Appl.*, vol. 9, no. 1, pp. 1–20, 2006.

[33] T. W. S. Chow and M. K. M. Rahman, "A new image classification technique using tree-structured regional features," *Neurocomputing*, vol. 70, no. 4–6, pp. 1040–1050.

[34] P. Salembier and L. Garrido, "Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval," *IEEE Trans. Image Process.*, vol. 9, no. 4, pp. 561–576, Apr. 2000.

[35] M. K. M. Rahman, W. P. Yang, T. W. S. Chow, and S. Wu, "A flexible multi-layer self-organizing map for generic processing of tree-structured data," *Pattern Recognit.*, vol. 40, no. 5, pp. 1406–1424, 2007.

[36] F. Kappe and B. Zaka, "Plagiarism—A survey," *J. Universal Comput. Sci.*, vol. 12, no. 8, pp. 1050–1084, 2006.

[37] S. Niezgoda and T. P. Way, "SNITCH: A Software tool for detecting cut and paste plagiarism," in *Proc. 37th Special Interest Group Comput. Sci. Edu. Tech. Symp.*, New York, Mar. 2006, pp. 51–55.

[38] S. M. Zu Eissen and B. Stein, "Intrinsic plagiarism detection," in *Proc. 28th Eur. Conf. IR Res.*, London, U.K., 2006, vol. 3936, pp. 565–569.

[39] N. Kang, A. Gelbukh, and S. Y. Han, "PPChecker: Plagiarism pattern checker in document copy detection," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2006, vol. 4188, pp. 661–667.

[40] N. Heintze, "Scalable document fingerprinting," in *Proc. 2nd USENIX Workshop Electron. Commerce*, Oakland, CA, Nov. 1996, pp. 18–21.

[41] P. Iyer and A. Singh, "Document similarity analysis for a plagiarism detection system," in *Proc. 2nd Indian Int. Conf. Artif. Intell.*, pp. 2534–2544.

[42] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[43] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 32, no. 4, pp. 431–443, 1996.

[44] E. Høgh-Rasmussen, "BBTools—A Matlab toolbox for black-box computations," Neurobiol. Res. Unit, Copenhagen Univ. Hospital, Copenhagen, Denmark, 2005 [Online]. Available: http://nru.dk/software/bbtools/

Tommy W. S. Chow received the B.Sc. (1st Hons) degree and the Ph.D. degree from the Department of Electrical and Electronic Engineering, University of Sunderland, Sunderland, U.K.

Currently, he is the Professor in the Department of Electronic Engineering, City University of Hong Kong. He has been working on different consultancy projects with the Mass Transit Railway, Kowloon-Canton Railway Corporation, Hong Kong. He has also conducted other collaborative projects with the Kong Electric Co., Ltd., the MTR Hong Kong, and Observatory Hong Kong on the application of neural networks for machine fault detection and forecasting. He is an author and coauthor of over 130 journal articles related to his research, five book chapters, and one book. His main research has been in the area of neural networks, machine learning, pattern recognition, and fault diagnosis.

Dr. Chow received the Best Paper Award at the 2002 IEEE Industrial Electronics Society Annual Meeting in Seville, Spain. He is the Associate Editor of *Pattern Analysis and Applications*, and the *International Journal of Information Technology*.

M. K. M. Rahman received the B.S. degree in electrical and electronic engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2001, and the Ph.D. degree in electronic engineering from City University of Hong Kong, Hong Kong, in 2007.

Currently, he is an Assistant Professor in the Department of Electrical and Electronic Engineering, United International University, Dhaka, Bangladesh. He has been engaged in research activities in the field of pattern recognition since 2002. He published a number of papers in recognized international journals. His research interests include pattern recognition and neural networks. Based on neural networks, he developed a number of applications such as image retrieval and classification, face identification.