SEMISTRUCTURED AND STRUCTURED DATA MANIPULATION

BY

KUO YIN-HUNG

SUPERVISED BY :

PROF. WONG MAN-HON

A THESIS SUBMITTED IN PARITAL FULFILMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF PHILOSOPHY

IN

COMPUTER SCIENCE & ENGINEERING

©THE CHINESE UNIVERSITY OF HONG KONG

MARCH, 2001

論文題目：半結構化與結構化數據之處理

作者：郭彥虹

學校：香港中文大學

學系：計算機科學與工程學系

修讀學位：哲學碩士

摘要：

基於萬維網(World Wide Web)的普及化，現在越來越多人將他們的資料存放在萬維網上以供存取。只可惜現在萬維網上的資料均是分散式的(decentralized)，因而導至存放在萬維網上的資料沒有一個完整的結構。而這點也是使人未能針對萬維網上的數據作出進一步處理的主因，亦因而導至很多其他的問題，例如有很多內容相近的資料卻因結構上的分別而難於搜尋，而內容相近的資料亦會因結構不同而被重覆存放在萬維網上。

雖然萬維網上的文件並沒有一個很完整的結構，但因萬維網上的所有文件也必須依據 HTML 的基本定義，所以我們可以跟據 HTML 的基本定義，將萬維網上的文件轉化為物件交換模型(Object Exchange Model)的數據(OEM data)。物件交換模型主要是用作處理半結構化的數據。而當萬維網上的文件轉換為物件交換模型的數據後，那些資料便可作出進一步的處理(如分類、聚類、索引等)，因而簡化了整個搜尋程序。而我們將根據被轉換成物件交換模型之數據提出一些新的運算法則，用以將萬維網的文件分類及聚類。

除此以外，我們亦會在這篇論文中討論將傳統的向量空間索引及聚類技巧(Vector-space Indexing and Clustering Techniques)合而為一而藉此改善傳統索引技巧的表現。現有的向量空間索引技巧存在著很多未能解決的問題，例如當向量空間的維數增加而起的表現退化問題(The problem of Dimensionality Curse)和當索引結構內的數據增加而引起的表現退化問題。聚類可以提供有關數據點及其附近數據點的資料，而藉著這些資料，對計算機需求較大的 knn-搜尋可以轉換成一個普通的定距搜尋。我們會在這篇論文的實驗部份中證明我們所建議的索引結構(稱為 IR-Tree)將比其他向量空間索引技巧有更好的表現。此外，我們亦會將這個建議引用到計量空間(Metric Space)上。

本論文的主要貢獻如下：(1) 提出一個跟據嵌在萬維網文件內的超連結和文件語義而訂定的相似性定義(similarity definition)；(2) 建議一個將萬維網分類和集成的運算法則；(3) 提出一個新的字典結構用作識別相關字和相關文件；(4) 提出一個用作抽取「普遍結構」(general structure)及減低在不同物件交換模型的數據

結構之差異的運算法則。此外，我們亦會 (5) 提出一個跟據現有的索引結構及聚類結構而成的新索引結構，並證明我們建議的索引結構能改善過往相類結構的表現。

Semistructured and Structured Data Manipulation

submitted by

KUO Yin-Hung

for the degree of Master of Philosophy

at the Chinese University of Hong Kong

Abstract

There are more and more people who put their information on the World Wide Web (WWW). However, the WWW grows in a decentralized process and the documents placed in the WWW are lack of logical organization. This makes the further manipulation of the data placed on the WWW a difficult task. There are several drawbacks concerning this problem, for example, it is difficult to retrieve similar or closely-related information, and the data placed on the WWW will be duplicated, resulting in the redundancy of resources.

Although web documents are of little logical organization, they should follow the basic definition of HTML (Hypertext Markup Language). Based on the basic definition from HTML, a web document can be transformed into an Object Exchange Model (OEM) data. OEM is designed for manipulating semi-structured data. Once the web documents are transformed to the same data model, they can be further manipulated, like classification, clustering and indexing for the ease of further retrieval of web documents. In this thesis, We will propose algorithms, based on the OEM data transformed, to classify and integrate web documents.

In this thesis, we will also give a discussion on the integration on vector space indexing and clustering techniques in order to enhance the performance of indexing. There are many problems associated with the existing vector space indexing techniques, like the dimensionality curse and the performance degradation upon the addition of data entries. Clustering provides information about its closely located data points. With the aid of clustering information, the expensive knn-search can be transformed into a range search. We will show in the experiments that IR-Tree, the proposed indexing structure, out-performs other spatial indexing algorithms under a highly-clustered environment. We will also try to extend the idea to indexing structures for metric space.

The major contributions of this thesis are : (1) propose a similarity definition based on the hyperlinks embedded in the web documents and the document semantics; (2) propose an algorithm for the classification and integration of related web documents into the same subset; (3) propose a dictionary data structure for identifying related words and documents; and (4) then propose an algorithm for retrieving the "general structure" and reducing the variation of the structures on different OEM records. Moreover, we will also propose a new indexing structure based on existing clustering and indexing techniques for which the performance can be improved. We will show in the corresponding experimental section that my proposed algorithms can successfully classify and integrate related documents and reduce the variations on the data structures generated from sets of semi-structured web documents.

Acknowledgments

First of all, I would like to thank Prof. Man-Hon Wong, my supervisor, for his guidance and patience. My research could not have been done reasonably without his insightful advice. For the past two years, he gave me encouragement, support, and guidance on my paper (published in PRICAI 2000 Text and Web Mining) and my thesis. It would not be possible to get my paper published without him.

My great gratitude goes to Prof. Ada Wai-Chee Fu and Prof. Ho-Fung Leung, who marked my term paper and gave me valuable suggestions.

My gratitude goes to the Department of Computer Science & Engineering, CUHK. It provides the best equipment and office environment required for high quality research to our students.

Finally, I thank to my fellow colleagues, who helped me in solving the programming, computer and official problems, and enlightened me to the new research ideas. They are (not in order) Polly Po-Man Wan, Kam-Wing Chu, Wing-Kai Lam, Po-Shan Kam, Wai-Chiu Wong, Wai-Ching Wong, William Chen, Hing-Wing Chan, Sam Ma, Clara Law, Kevin Wong, Michael Fung, Vincent Cheung, Willis Chan, Men-Hin Yan, Ricky Wong and Yin-Ling Cheung. They gave me a happy and wonderful university life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World Wide Web (WWW) is a treasure of information in the sense that you can find any information you want on the web. There are billions of web documents on the WWW and thousands appearing each hour. However, the WWW is growing through a decentralized process, and documents in the web are lack of logical organization in the sense that similar or related documents are presented in totally different styles and have no inter-relation. This leads to the difficulty for the users and computers to access and process those web documents.

As the improvement on the Web technology, not only text, but multimedia data, like images, sounds and movies can also be found in various homepages. However, the majority of web documents are still in the form of plain text, or text with little multimedia data embedded in. If we consider only the textual data on the WWW, it can be considered as a large but un-structured database containing billions of web documents. Although there are so many web documents, we need not analyze all the web documents as a whole. Web documents can usually be divided into disjoint sets based on their document content. For example, web documents concerning automobiles have no or very little relation with documents about various programming languages. It is meaningless to compare the inter-relation between an automobile company homepage and http://java.sun.com/. Hence, some algorithms [10, 25] were proposed to partition the huge set of web documents into smaller subsets that group related documents together for further operations.

Once the huge set of documents are partitioned into smaller subsets, more operations can be done. If documents are partitioned into subsets of related documents, the documents can be organized in a hierarchical manner. Moreover, keywords can be extracted from the subset of documents for the representation of the subset. Features are extracted from all the documents in each subset and the set of features can be considered as a set of entries in a large database. Clustering and indexing can be done based on the set of entries extracted from the subset of documents and the retrieval of web documents can be enhanced.

The objective of the first three chapters of this thesis is to propose a methodology to process the set of web documents. Web documents are organized in a decentralized process and hence we will try to partition the set of web documents into meaningful subsets. Then the set of web documents in the same partition will be integrated and indexed. The final goal of this thesis is to extract meaningful information about the web documents from each subset and enhance the retrieval of web documents.

There are already many web document classification algorithms proposed, and many of them are applied on various search engines. However, the classification and integration algorithm is different from traditional algorithms in the sense that most of the existing algorithms classify web documents by using keywords extracted from the document set. There are also some other algorithms that extract a feature vector from each document and classify web documents based on the feature vectors extracted. However, there is no much algorithm proposed that considers the structures of the web documents upon the similarity computation. The consideration of the structure of web documents take an important role on the web document integration algorithm.

Instead of handling web documents, the later parts of this thesis describes another problem based on the domain of vector space. We will focus mainly on the integration of indexing and clustering techniques to improve the performance of searching from an indexing structure. There are already many indexing structures proposed for indexing vector space data. However there are many problems associated with Spatial Access Methods (SAMs). For example, the problem of dimensionality curse [5] associated with those SAMs cannot be solved efficiently. Moreover, as the data size grows, the performance of those SAMs will deteriorate. In the later part of this thesis, we will introduce an integration of clustering and indexing techniques, the IR-Tree. The proposed new indexing structure, IR-Tree, solve the problems of performance degradation upon the structural reorganization and the increase of dimensionality of data (dimensionality curse [5]). Experiments showed that the performance of IR-Tree out-performs other SAMs.

Then, we will try to extend the idea used in IR-Tree to improve the performance of indexing structures for metric space. Vector space is a subset of metric space. However, there are less research efforts placed on the indexing of metric space when compared with that placed on vector space indexing techniques. In the later two chapters of this thesis, we will discuss the algorithms in detail.

1.1 Web Document Classification

From chapter 2 to 5 of this thesis, we will discuss a web document classification methodology, together with a similarity definition. A web document is considered as a document containing self-described information, together with a set of hyperlinks embedded in the document that points and establishes a relation with other web documents. However, due to the semi-structured property of web documents [13], web documents should be transformed to a standardized data structure before comparing their content. Object Exchange Model (OEM) is used to manipulate semi-structured data. In this thesis, web documents will be transformed to an OEM data before comparison and further manipulation.

However, it is not feasible to compare the huge set of OEM data for similarity comparison. A web document, besides its content, also contains a set of hyperlinks that points to other web documents. These sets of hyperlinks can provide information about inter-relationship among web documents. For instance, a web document about Java programming language should contain a set of hyperlinks that point to other web documents about Java, or other object-oriented programming languages. Instead, no one will expect a web document about Java contains a hyperlink that points to an automobile company. We will first transform the set of web documents to a directed graph and retrieve a set of important web documents (center nodes) based on two definitions: *importance* and *reference*. Then based on the set of center nodes, the large set of web documents can be partitioned into smaller and related subsets. The details concerning the partitioning based on hyperlinks will be discussed in detail in Chapter 3.

Hyperlinks can help partitioning the web documents into smaller subsets, but it is insufficient. Document content, in addition to the hyperlinks embedded, should be used to classify web documents. Thus, after the coarse partitioning based on the hyperlinks, the documents within a subset should be refined such that all the documents within the subset should concern about the same, or a related topic. In Chapter 3, we will give a similarity definition based on the document content. Documents are difficult to compare with their similarity directly. Therefore, documents will be transformed into OEM data before comparison, and the similarity between the OEM data within a subset is computed based on the similarity definition given. A representative document will be retrieved from the subset of documents and the algorithm will also be discussed in Chapter 3.

1.2 Web Document Integration

Even after web documents are classified into different subsets, the structure of the OEM data within the same subset (the layout of web documents) is still varied. This leads to an unnecessary waste of storage for keeping the original structure of web documents. Moreover, due to the variety of web documents, further operations, like searching and indexing, are also hard to perform.

Therefore, in order to minimize the variation, a general structure will be generated for each subset of documents and the general structure is used for the manipulation of all the OEM data in the subset. The general structure is generated based on the OEM and all the OEM data within the same subset will be transformed following the general structure of their subset. The detail of the general structure generation and the transformation algorithm will be discussed in detail in Chapter 4.

1.3 Dictionary and Incremental Update

In order to increase the correctness of the similarity computed, a dictionary data structure is introduced. The dictionary data structure keeps the information of related keywords and the occurrence of each word in the corresponding OEM record. The dictionary helps locating the occurrence of keywords and speeds up the performance of keyword search. Moreover, the information of related keywords can also be used to increase the accuracy of similarity computed. The details of the dictionary will be discussed in Chapter 5.

However, as mentioned before, there are thousands of new web documents placed on the WWW. Thus, an incremental update algorithm is essential to keep the database updated. The incremental update of the dictionary, as well as the general structure and the web documents will also be discussed in the corresponding chapters.

1.4 IR-Tree

There are many spatial access methods (SAMs) which are proposed for spatial data indexing, such as R-Tree [26], SS-Tree [52], SR-Tree [34], X-Tree [6], etc. Most of them are variations of R-Tree, which partitions data into different regions of minimum bounding rectangles (MBRs). Some modifications on R-Tree ,like R*-Tree [4], alter the optimization criterion upon splitting of nodes and try to attain a better optimization on the index structure. Experimental results show that all these variations can improve the performance of the original R-Tree indexing structure.

However, there is a fundamental problem associated with R-Tree: the performance will deteriorate gradually when the number of points inserted in an R-Tree increases. As the number of points increases, the overlapping of the MBRs will increase, hence resulting in the accessing of extra and unnecessary nodes. Though there are some proposals to reduce the overlapping of nodes (like the Forced Reinsert in R*-Tree), the problem still exists.

The problem of dimensionality curse is also one of the main weaknesses of SAMs. By using X-Tree as an example, experiments show that the performance deteriorates as the dimension increases, and a search tends to touch all the data pages when the dimension of the data is higher than 10 [5]. This drawback is seldom handled in SAMs and so it seems those general SAMs are not suitable for high dimensional data.

Motivated by the drawbacks above, we propose an enhancement on the R-Tree methods based on a clustering algorithm Incremental-DBSCAN [22]. Incremental-DBSCAN is based on DBSCAN [21], a clustering algorithm which searches for the neighbors based on the pre-defined distance *Eps* and *MinPts* as the basis of cluster generation. Upon insertion, the vector will be added to the corresponding cluster(s). The resultant clusters will take an important role in the performance improvement on the searching of relevant points.

In this thesis, we will introduce a new indexing structure, *IR-Tree*, which is based on the combination of R*-Tree and Incremental-DBSCAN. The difference between IR-Tree and other SAMs is that the clustering information will be utilized for the searching of elements. used in R*-Tree, each vector will be clustered into corresponding clusters. The clusters generated will be used to improve the performance of k nearest neighbor search (knn-search for short).

1.5 Thesis Overview

After giving the introduction on my works, a global picture on my work is given in this section. It is difficult to index the WWW. However, based on our approach, web documents can be indexed. Web documents are transformed and integrated into the same data structure based on the algorithm proposed. Once the set of web documents are integrated together, they can be grouped based on the "general structure" generated. Then the set of general structures can be transformed into a set of multi-dimensional vectors. The set of multi-dimensional vectors can then be indexed by using IR-Tree introduced in the later chapter of this thesis.

In this chapter, we have introduced the contributions of this thesis, which are web document classification and integration as well as an integration of indexing and clustering technique. In Chapter 2, we will give some backgrounds on the problems we did, especially some techniques on web document classification, integration as well as the indexing techniques on vector space. The web document classification algorithm as well as the definitions on different types of documents will be discussed in Chapter 3. In Chapter 3, transformation algorithm from a web document to an OEM record, similarity definitions and the representative selection algorithm will also be presented. The web document integration algorithm is discussed in Chapter 4. The details about the dictionary data structure will be given in Chapter 5. The details of IR-Tree and the suggested new indexing structure based on the integration of M-Tree and Incremental-DBSCAN are discussed in Chapters 6 and 7 respectively. In addition, performance analysis and experiments are carried out throughout this thesis to show the performances of our proposed algorithms. We conclude our works in Chapter 8.

Chapter 2

Related Works

In this Chapter, we will first discuss some issues on web document classification, web document integration and indexing. Throughout the first part of the thesis, we use Object Exchange Model (OEM) for the manipulation of web documents. Web documents are characterized by its semi-structured property. Thus, we will also discuss the issues concerning OEM in this chapter.

2.1 Semi-structured Data and OEM

OEM is mainly designed for the manipulation of the semi-structured data [13]. Thus, before discussing about the OEM, we will give an introduction on the semi-structured data.

2.1.1 Semi-structured Data

Traditionally, data is manipulated in the form of relational data or object-oriented data. However, some of the data may not appear in the form that can be constrained by a schema. The most immediate example of such data is the WWW data. Ahthough researchers want to treat the WWW as a large database, there is no standard model to describe the structure of the WWW documents. Hence, it is difficult to formulate the queries.

Moreover, there is no all-embracing data model that can be used to model all the data. Without such a data model, it is difficult to build a software that can convert between two desperate models. Besides, for traditional data models, it is difficult for the users to formulate queries which may have opaque terminology and the rationale for the design. Thus, the author of [13] proposed the idea of semi-structured data, which use a graph-like or tree-like structure to represent the data.

2.1.2 Object Exchange Model

OEM was proposed in [13] for the representation of semi-structured data. OEM is a labeled tree with a label associated with each edge, which can be formulated as follows:

$$typelabel = int|string|...|symbol$$

$$typetree = set(label \times tree)$$

The first line describes a tagged union or variant, while the second line mentions that a tree is a set of label/tree pair. Some variations on the basic OEM are proposed, where internal nodes are empty and symbols are placed in leave nodes.

$$typebase = int|string|...$$

$$typetree = base|set(symbol \times tree)$$

This is not the only format of OEM. There are some possible variations on the OEM tree structure, and the structure above is only one of them. In this model, there is no meaningful data stored in the internal nodes of the tree, and all the data are stored in leaf nodes. There is no ordering on the edges. Please refer to [13] for more detail on about OEM.

OEM is the data model proposed for manipulation and representation of data with no rigid schema. This is particularly useful for representing web documents, which is characterized by having no rigid schema. In chapter 3, 4 and 5, an algorithm will be given on transforming web documents into OEM and generating a "global structure" in OEM for each subset of documents which can represent all the data from documents in the subset.

As a remark, relational and object-oriented database can be transformed into this model. However, for the case of object-oriented data, the issue of object identity has to be handled with great care. The transformation from other databases to semi-structured database is not unique, and different results for different transformation algorithms may be resulted.

2.2 Related Work on Web Document Partitioning

In the first part of the thesis, we will try to partition the set of web documents based on hyperlinks and the document semantics. Most traditional web document categorization techniques partition web documents based on the document content. However, in this thesis, we will try to use the hyperlinks embedded in web documents and the document semantics for the partitioning. In this section, we will briefly introduce some related works on web document partitioning as well as the utilization of hyperlinks in partitioning web documents.

2.2.1 Retrieval of Authoritatives

The analysis of hyperlinks in web documents was proposed in [35, 25]. HITS [25] is the application based on [35]. In [35, 25], the authors gather a set of related documents from different search engines and transforms the set of hyperlinked documents as a directed graph $G=(V,E)$: the nodes correspond to the pages, and a directed edge $(p,q) \in E$ means the existence of a link from p to q. Then the author defines two types of documents, namely *authorities* and *hubs*. Authoritative pages are documents which have a high *in-degree* (the number of links pointing to a document), while hubs are those documents which point to multiple authorities. Authoritative pages and hubs should have a mutually reinforcing relationship: a good hub should point to many authoritative pages, while a good authority should point to many hubs. Moreover, in this paper, *unrelated page* is also defined, which is a document with high in-degree and very low out-degree, like some advertising documents. By computing their in-degree and *out-degree* (the number of links points to others from a document) iteratively and eliminating irrelevant ones, authoritatives and hub pages can be identified.

However, there is a problem associated with the algorithm. The algorithm works well if the query is specific. For example, if query on "Java" is performed, the document http://java.sun.com/ will be retrieved easily. However, if the query is not specific, it may not be possible for the algorithm to return the authoritative pages. For example, if a user searches for "University", a huge set of documents will be returned. They should be grouped into different subsets. However, by using this algorithm, only one or two authoritative pages will be retrieved. This is not desirable.

2.2.2 Document Categorization Methodology

There are many web document categorization methods proposed. HyPursuit [51] uses semantic information embedded in the hyperlink structure and document content to cluster web documents. Authors in [51] first defined the similarity based on the content-links. The similarity definition counts on the number of *common ancestors* and *common descendents* between two documents. Then the authors define the similarity based on the content of documents that based on the *term frequency* and the *weight contribution.* Clusters are formed based on the two similarity definitions.

Instead of browsing the entire set of web documents or gathering search results from various search engines, Bookmark Organizer (BO) [40] takes a local view on the web documents that the user browses. BO clusters collections of web documents which is in the form of bookmarks by using hierarchical clustering techniques. It works on a semi-automatic way: it combines manual and automatic organization on the bookmarks to enhance the accuracy on categorization.

WAKNN (Weight Adjusted k-Nearest Neighbor) classification algorithm [28] is based on k-NN classification paradigm. [19]. The weights of features in WAKNN are learned iteratively and only the weights of feature with the most improvement in the objective function will be updated. PEBLS [17] and VSM [39] are also k-NN classification algorithms which applies the Modified Value Difference Metric (MVDM) [17] and conjunction gradient optimization technique [47] for the learning and update of the feature weights.

While WAKNN learns the weight of features by an iterative approach, the authors in [36] use Bayesian classifiers [33] in the feature selection process. The authors propose an approach that utilize the hierarchy in topics to reduce the size of feature set, and use Bayesian classification methodology to classify documents. Besides [36], AutoClass [14, 48] also applies Bayesian classification methodology for document classification.

There are also some other approaches such as those using association rule hypergraphs [29, 42] and mobile agents [55]. However, most document classification methodologies, which are mainly distance- or probability-based, are inadequate in the sense that they require a large number of pre-inputted features for learning (probabilistic approaches) or improper distance measure due to variations of documents in length and wordings (for distance based approaches). Hyperlink is a common feature for all web documents. By analyzing the set of hyperlinks in a document, inter-relations among documents can be observed, without learning of special features required. However, there is not much research for web document classification that utilize the set of hyperlinks in web documents. In Chapter 3 of this thesis, we will discuss the web document classification algorithm by hyperlinks in detail.

2.3 Semi-structured Data Indexing

Together with the introduction of semi-structured data, indexing methodologies for semi-structured data are also introduced. The most well-known one is Lore [41], a DBMS designed for the storage and representation of semi-structured data. There are also some algorithms which manipulate semi-structured data.

2.3.1 Lore

As mentioned before, Lore (Lightweight Object REpository) [41] is a DBMS designed for the storage and representation of semi-structured data. It uses OEM to represent the data. Together with the query language specific for Lore, Lorel [1], Lore can be used to store, query and index semi-structured data.

2.3.2 Tsimmis

Tsimmis (The Standard IBM Manager of Multiple Information Sources) [27, 44] is a project in IBM that provides an architecture and tools for accessing multiple heterogeneous information sources, which will translate the data into Object-Exchange Model (OEM) [27, 1].

2.3.3 Other Algorithms

There are some papers studying on matching the structure of semi-structured data [7], while some others try to store the semi-structured data in form of relations [18]. However, they just concentrate on the original structure of data. If the structure deviate too much, their algorithms cannot be applied. Moreover, those algorithms usually require extra information or well-defined and standardized structure. In [3], a wrapper is implemented to automatically wrap HTMLs into structures and perform searching. In [56], the authors propose an algorithm which retrieve the n most related documents from a set of search engines based on the representatives generated by the database search engine. Then documents are ranked following their relevance.

2.4 Related Work on SAMs

There are many Spatial Access Methods (SAMs) proposed in order to provide a fast and robust searching on vector space data. Most of them are derivations of R-Tree [26]. In this section, we will give an introduction on R-Tree and its derivatives, as well as some other indexing methodologies.

2.4.1 R-Tree and R*-Tree

Among various spatial access methods, R-tree [26] is the most common and fundamental one. The basis of R-tree is to build a multi-dimensional index structure by packing data points into rectangles. It is a height-balanced tree, with leaves pointing to various data objects. Both leaf nodes and non-leaf nodes have the minimum bounding rectangles (MBR), with MBR on leaf nodes bounding all its data points and that of non-leaf nodes bounding all its children MBRs. That means, an R-tree is a hierarchy of bounding rectangles.

However, R-tree suffers from the performance deterioration when more and more data are inserted into the tree. Upon splitting, R-tree is based on a heuristic optimization, minimizing the enclosing area of rectangles. Experiments proved that the performance of R-tree drops when more data are added into it. Hence, a variant of R-tree, R*-tree [4] is proposed.

The main difference between R-tree and R*-tree is that the latter employs the forced reinsert mechanism on inserting data and splitting nodes. In R*-tree, when a node overflows, parts of its entries will be re-inserted. This leads to an improvement on the organization of points and nodes.

2.4.2 SS-Tree and SR-Tree

Instead of using MBRs that R-tree and R*-tree do, there are some variations of R-tree which use a different hierarchy. SS-tree [52] is one of these examples. In SS-tree, data are bounded in spheres rather than rectangles. The nesting property of R-tree is retained in SS-tree, i.e. the bounding spheres of all children nodes are nested in their parent sphere. The center of a sphere is the centroid of all points in its subtree. It also allows the tree to divide points into isotropic neighborhoods during insertion and splitting. Upon insertion of a new point, the subtree with the closest centroid will be selected and propagates downwards. If the node is full, the variance for all the children with the centroid is calculated on each dimension, and split on the dimension with greatest variance.

There are several advantages for using bounding spheres instead of using rectangles in an index structure. Firstly, using bounding spheres can improve the performance of nearest neighbor search. As the sphere bound all its children in a specific radius, it is easier to compute the nearest neighbor and prune away irrelevant members using the triangular inequality of Euclidean distance. Moreover, the storage space can be reduced as only the centroid and covering radius are stored in a node for bounding sphere, while 2**dimensionality* number of variables (the bounding rectangle) have to be stored for each node in an R-Tree.

However, there are also some drawbacks for using hyper-spheres rather than hyper-rectangles in an indexing structure. By using bounding spheres, the covering area is much larger than that of using bounding rectangles. This leads to an increase in the number of node access when a range search or an insertion is performed.

SR-tree [34], considering the pros and cons of rectangles and spheres, uses both spheres and rectangles to index data. A leaf stores the data point, while a non-leaf node uses a bounding rectangle and sphere to indicate its bounding area, which is the intersection of two areas. The centroid of bounding sphere is computed by taking the average of all children on all dimensions; the radius of bounding sphere is the minimal distance which can bound all the children rectangles and spheres. Experiments show that SR-tree outperforms SS-tree and R^*-tree upon nearest neighbor queries.

2.4.3 TV-Tree and X-Tree

While SS-Tree and SR-Tree improve the performance of the R*-Tree by applying hyper-spheres instead of hyper-rectangles, there are some other SAMs which try to enhance the performance by adding and altering some features of the R*-Tree. TV-Tree [38] and X-Tree [6] are the examples of such SAMs. In TV-Tree, the dimensionality of feature vectors is reduced and only the active dimensions are kept for indexing high dimensional data. All the dimensions are sorted according to their importance and only the first few active dimensions are kept as the index. The active dimensions may shift if for most of the feature vectors in a sub-tree, the coordinates for the most important active dimension are the same. That dimension is then set inactive and the next important dimension is set to be the new active dimension. However, as mentioned in [52, 34], the effectiveness of TV-Tree depends much on the nature of the dataset, in which the feature vectors should allow the shift of active dimensions. Hence, the performance of the TV-Tree may not be as good as expected for real data.

Instead of reducing the dimensionality of feature vectors as that in [38] does, X-Tree [6] tries to improve the performance of R*-Tree by employing the overlap-free split and the use of supernode. In X-Tree, the split of nodes is ensured to be overlap-free, so that the performance of point queries can be improved. However, if the overlap-free split is not possible, a supernode, which is an oversized node, is introduced. The idea of the supernode is to circumvent the overlap area so that the I/O throughput for reading and writing nodes can be reduced.

2.5 Clustering Algorithms

SAMs use hyper-rectangles or hyper-spheres for spatial management and the hierarchy between the parents and its children is clear. For example, in SS-tree or SR-tree, a node is represented in a hyper-sphere (or a combination of hyper-rectangles and hyper-spheres). So, the relation between children and its parent is obvious: all children are bounded within the radius from the centroid of the bounding sphere of its parent node. However, it is difficult for those indexing structures to maintain the inter-relations among data points.

Clustering [37] is one of the numerous methods for maintaining information among related points, which is to group closely related data into corresponding groups. Clustering algorithms can be divided into three different categories, namely model-based and optimization-based, density-based and hybrid approaches. Model-based and optimization-based approaches like K-Means [24] and CLARANS [43] try to retrieve and assign a set of cluster centers (like medoids in [43]) and use the optimization criteria specified in the corresponding algorithms for cluster assignment. For density-based approaches, clusters are identified if the density of the data is larger than the threshold specified for the corresponding algorithm. DBSCAN [21] identifies clusters in form of a sphere, while STING [50] and Hierarchical Grid Clustering [45] partition the data space into a grid structure and there is no shape specified for clusters in DBCLASD [54]. Some transformation techniques are also applied for cluster identification. The authors of WaveCluster [46] apply wavelet transform for the identification of clusters. For hybrid approaches, like BIRCH [57] and CLIQUE [2], a coarse clustering on the dataset will be performed first and further operations on the set of coarse clusters (like merging of closely-located clusters) are done. In the later parts of this section, we will discuss more on one of the clustering algorithms, DBSCAN [21], and its variant, Incremental-DBSCAN [22].

2.5.1 DBSCAN and Incremental-DBSCAN

Incremental-DBSCAN [22] is originated from the clustering algorithm *DBSCAN* [21]. In DBSCAN, every data $d = d_1, d_2, ..., d_n$ is scanned. If the number of points bounded in the "sphere" with d as center is larger than a threshold *MinPts* within a specified radius *Eps*, d is a cluster center and a cluster is identified. For those data points which have insufficient support within the bounding radius, they are regarded as noise. By using SAMs like R*-tree, the searching of all density-reachable objects is performed as a range query and hence by recursively searching, clusters can be found.

DBSCAN defines the relation *directly density-reachable* between two objects: if an object p has more than *MinPts* bounded within the radius *Eps* and q is one of its bounded point, then q is said to be *directly density-reachable*, or *ddr* from p. If two centers p and q are directly density-reachable with each other, the two clusters will be combined into one and all density-connected clusters will be combined into one. For two objects p and q, if there exists a set of objects $p_1, p_2, ..., p_n$ such that p is *ddr* from p_1, p_n is *ddr* from q, and p_i is *ddr* from p_{i+1}, p and q are said to be density-connected. Clusters are defined as the set of all density-connected objects. All objects not included in clusters are regarded as noise.

However, DBSCAN can only be applied on a static database. Incremental-DBSCAN modifies the algorithm of DBSCAN and make it applicable to a dynamic database. It first defines which object(s) (clusters or data points) will be affected during insertion (or deletion) of a cluster. Different scenarios and their corresponding handlings to the clusters affected are also discussed in detail in [22]. With this modification, the database need not be scanned and can handle the clustering problem dynamically.

Chapter 3

Web Document Classification

There are billions of web documents in the WWW, and thousands of new documents coming every day. Due to the enormous size and the dynamic behavior of the web documents, it is impossible to process them as a whole. Moreover, as mentioned before, web documents are unstructured so that it is difficult to estimate the similarity between web documents. In this Chapter, we will discuss the details of web document partitioning by hyperlinks. By considering the web documents as nodes and hyperlinks in the document as edges, the set of web documents can be transformed to a graph. The partitioning of web documents is based on the graph generated.

3.1 Basic Definitions

Before the presentation of the partitioning algorithm, we will give a set of basic definitions concerning the graph transformed and the set of nodes (web documents). Web documents usually contain a set of hyperlinks. By considering the sets of web documents as nodes and hyperlinks as directed edges, any set of web pages W can be transformed into a directed graph $G=(V,E)$: a web page corresponds to a node in W and a directed edge $n(i,j)$ where $i, j \in V$ means there is a hyperlink from document i to document j. This definition is the same as that used in [35]. The following is the definition of a directed link in such a graph G.

Definition 3.1 (Directed Link) *For a graph G=(V,E),* $i, j \in V$, $n(i,j) \in E$ *= 1 if there is a directed link from node i to node j, or 0 otherwise.*

In this chapter, we want to propose an algorithm that performs a preliminary partitioning on the enormous set of web documents into smaller partitions. It is easier to handle a smaller subset of documents instead of handling all the documents at a time. However, the partitioning of web documents should gather related documents together instead of just dividing the set of documents into random subsets although the partitioning is a preliminary one. Moreover, the algorithm should be fast and simple in the sense that it should be capable of handling a large set of documents in a short time. Hence, we propose to use hyperlinks in web pages for partitioning.

Before presenting the algorithm, we have to classify web documents into different types. In this chapter, we assume that hyperlinks in web pages should be relevant to the topic that the document describes. However, not all hyperlinks in a web page is related to the specific topic that the document is about. Some links in a web document are related, while some of them are just link to some common sites like http://www.yahoo.com/. Some web documents even contain many hyperlinks that link to some totally unrelated documents, as some personal homepages do. Thus, before partitioning web documents, we should classify different class of web documents according to the following definitions:

Definition 3.2 (Importance) *The importance of a web document i is defined* *as*

$$\sum_{j \in V} n(i,j) \times n(j,i) / \sum_{j \in V} n(i,j)$$

Definition 3.3 (Reference) *The reference of a web document i is defined as*

$$\sum_{j \in V} n(j,i) / \sum_{j \in V} n(i,j) + n(j,i)$$

Importance of a document defines how important the document is among the set of documents by considering a document's *in-link* and *out-link*. The importance of document i defines the number of documents that have links to document i and document i has links to those documents. If *importance* is large, it means the document is an important document which is of high inter-relation with other documents. The *importance* can be used to identify the important documents (which should have links pointing to other related documents and being pointed by those documents) from irrevalent ones (which many documents have hyperlinks pointing to those documents but they do not have links pointing back), like the set of search engine pages.

Reference of a document defines how many documents have hyperlinks pointing to the document. In other words, the *reference* of a document shows how many documents take that document as an reference. For those documents which have high *reference* value, it may also be an important document (with high *importance*) or it is a commonly-referenced page (with low *importance*) like http://www.yahoo.com. Based on definitions 3.2 and 3.3, documents can be classified into different types.

As mentioned before, documents that discuss about a programming language probably do not contain a hyperlink to an automobile company. Thus, the hypergraph V should be able to partition into set of subgraphs. We define documents within a subset of documents as *related documents* or *related nodes.* Within the set of related nodes, two different types of nodes can be identified: *center nodes* and *terminal nodes. Center nodes* are those which are being pointed to by many documents in the subset; while *terminal nodes* are nodes that have no directed edges to other nodes except those which have edges pointing to that node. Besides *related nodes*, there are some nodes in the hypergraph that link up several subsets but not on a specific topic. We name this type of nodes as *unrelated nodes.* The following are the definitions of nodes based on definitions 3.2 and 3.3:

Definition 3.4 (Center nodes) *For a node $i \in V$, if importance $\geq \alpha_1$ and reference $\geq \alpha_2$, the node is considered as a center node.*

Definition 3.5 (Unrelated nodes) *For a node $i \in V$, if importance $\leq \delta_1$ or reference $\leq \delta_2$, and n(i,j) are not pointing to the set of center nodes in a partitioned subgraph, the node is considered as an unrelated node.*

Definition 3.6 (Terminal nodes) *For each node $i \in V$, if $\sum_{j \in V} n(i,j) < \theta$, and $\sum_{j \in V} n(i,j) = \sum_{j \in V} n(i,j) \times n(j,i)$, the node is considered as a terminal node.*

Figure 3.1 shows graphically the various types of nodes. In (a), the set of nodes are related together; while in (b), there is an *unrelated node* that links up two different subset of nodes, which are of no inter-relation. In (c) and (d) of figure 3.1, the node(s) which is/are hollow represent the *center node* and *terminal nodes* respectively. The *center nodes* can be used as seeds for the generation of

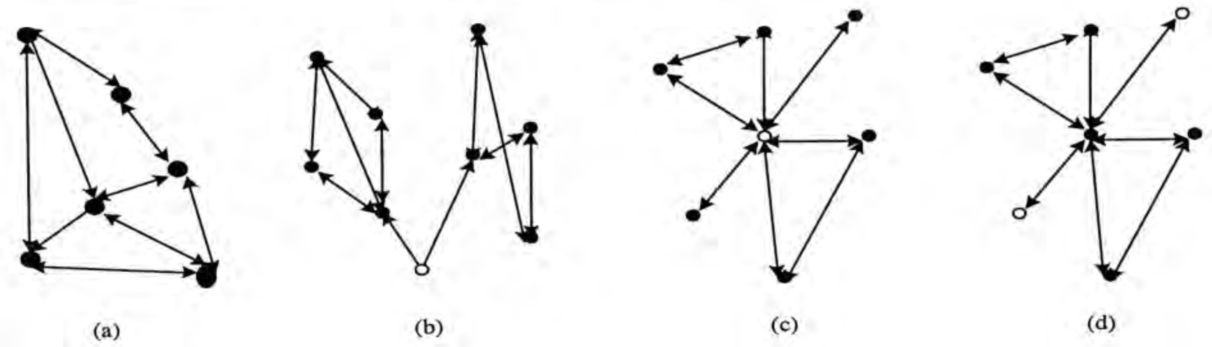


Figure 3.1: Graphs showing (a) related nodes, (b) unrelated nodes, (c) center nodes and (d) terminal nodes

subsets. The set of parameters can be tuned so that the partitioning of documents can be done more efficiently and accurately. If the set of nodes are of a related topic (gathered from a set of search engine's result), parameters δ_1 and δ_2 can be set larger, so that less documents will be eliminated. However, if the set of documents are just gathered by random, α_1 and α_2 should be set smaller, which is for a coarse classification first, while preventing from pruning away related documents.

By taking the set of center nodes as seeds, the set of related documents can be generated in an *a-priori* manner. The set of nodes that center nodes have edges pointing to are put in the same set. If nodes added into a subset are not *terminal nodes* nor *unrelated nodes*, the set of nodes that those newly-added nodes point to will be added to the subset too. The algorithm terminates when an *unrelated node* or a *terminal node* is encountered for all edges. If there are more than one *center node* (or *related nodes* in different subsets) that have edges to a node (not an *unrelated node*), those subsets should be combined into one.

Then, based on the set of center nodes in each subset of documents, *potential representatives* can be identified. *Potential representatives* can be used to represent the subset of documents. In this stage, the set of *representatives* is only *potential* because re-confirmation by using the algorithms proposed in later sections has to be done.

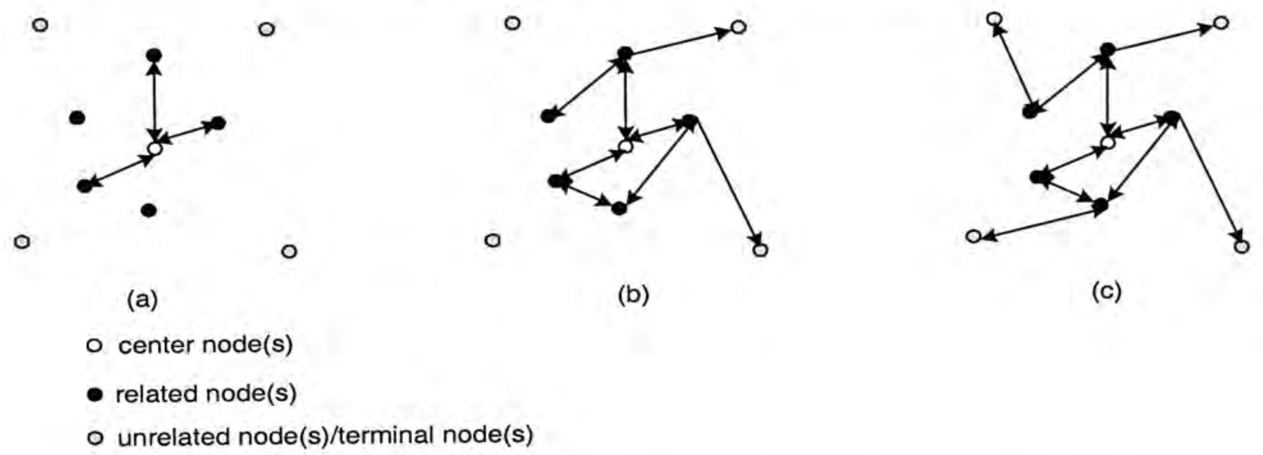


Figure 3.2: Generation of sub-graphs from the hypergraph of web documents

The set of *potential representatives* is the minimal set of *center nodes* that point to the largest number of documents in the subset. Greedy algorithm [20] is employed that the center node with the largest number of *out-links* to the documents in its subset is selected. If all the nodes that a center node points to are being pointed by the set of nodes in the *potential representatives*, the node is ignored from the set of *potential representatives*.

3.2 Similarity Computation

In this section, we shift the focus of our computation from a global view to a local one, which will focus on the similarity between documents within a set of documents and retrieve the most important document (*the representative*) in a set. However, before computing the similarity, all the web documents should be transformed to the same data model for similarity computation. Web documents are characterized by its *unstructured* property. It is difficult to compare their similarity if the documents are of different formats. In this thesis, we employ the *Object Exchange Model (OEM)* [13] as the data model for the manipulation of web documents.

3.2.1 Structural Transformation

As mentioned before, web documents are characterized by its *unstructured* property. Though web documents are *unstructured*, the construction of web documents should follow the basic definitions on Hypertext Markup Language [8]. In this thesis, we define a properly-formatted web document as follows:

1. It should follow the syntax of HTML strictly: started with <HTML> tag, body content enclosed by <BODY> and </BODY> tags, etc..
2. "Structure" should contain in an HTML document, which have headings, sub-headings and content using different tags enclosed in a hierarchical way.
3. A document should exist some hyperlinks that link to other web documents, and vice versa (it is not an isolated page).

Basically, a properly-formatted web document should be able to be transformed into an OEM data record. For example, an HTML document as shown in figure 3.3 can be transformed into an OEM record in figure 3.4 by following the hierarchy of tags in an HTML document.

Among the set of tags that an HTML document uses, some of them can be used as the identification of the document structure and the hierarchy of the document content. The set of tags, which can represent the importance of some words, like title and topic headings in a document, are named **structural tags**. Table 3.1 shows some of the *structural tags* and their corresponding *id* values.

Each tag in the set of *structural tags* are assigned with an id corresponding to its level in the hierarchy of an HTML document. For example, if <H1> is assigned with an id of 3, <H2> should be assigned with an id larger than 3, which indicates <H1> is on a higher level of hierarchy than that of <H2>. Based on

```
<HTML>
<TITLE>
     A simple HTML page
</TITLE>
<BODY>
<H1>
     How to write an HTML page?
</H1>
     It is not a difficult task to write an HTML page.
<H2>
     HTML Tags
</H2>
...
...
</BODY>
</HTML>
```

Figure 3.3: A sample HTML page

Structural Tags	ID
H1	1
H2	2
H3	3
BOLD/ITALIC	7
TABLE/UL/OL/DL	8

Table 3.1: Some structural tags and its corresponding id

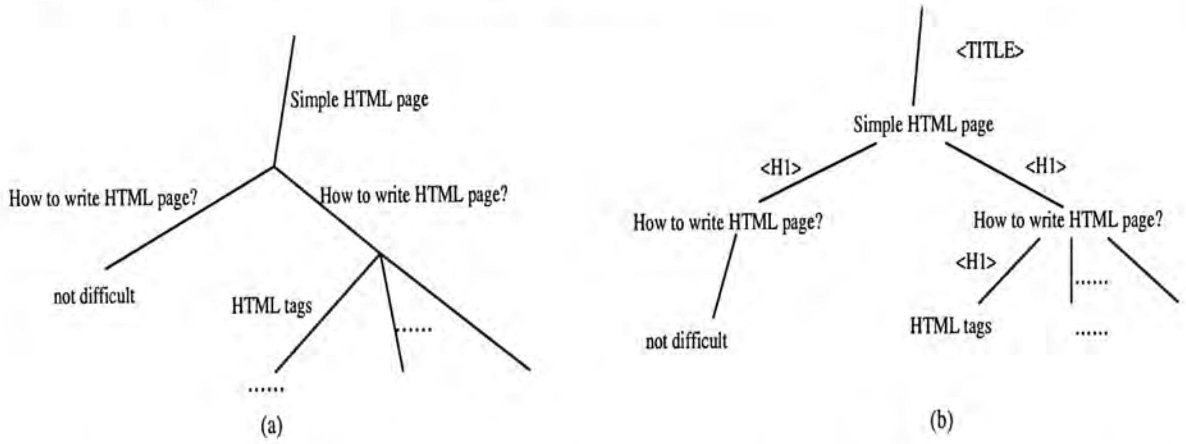


Figure 3.4: (a) The transformation of an HTML into OEM format (b) The corresponding preliminary OEM format

the set of *structural tags*, a web document will be converted into a set of nodes organized in a hierarchical way. Conceptually, the transformation algorithm converts a web document into a preliminary OEM record as shown in figure 3.4b with tags as edge labels and section headings as internal node content. Then by promoting the internal node content as edge label and removing the original edge label which is originally a tag, the OEM record in figure 3.4(a) can be generated.

After the set of documents are transformed into OEM, the similarity between documents can be computed. However, before the computation of similarity, we have to define the similarity between two documents. In this thesis, the similarity definition is defined on the OEM of the set of web documents instead of the content of two documents based on *feature vectors*. We will define the similarity of two documents under three aspects, namely *node similarity, edge label similarity* and *structural similarity*. The overall similarity between the two documents is the linear combination of the three similarities.

3.2.2 Node Similarity

A node in an OEM record transformed from a web page is a set of words extracted from the content of the corresponding document. In this chapter, the computation of node similarity is based on overlap of keywords. Basically, we define the *individual node similarity* between two nodes first, and then the *overall node similarity.*

Individual node similarity between two nodes i and j $nns(i,j)$=(number of words in i that match with j)/(number of words in i)

For example, if the content of node i is (a b c d) and that in node j is (c d e f), $nns(i,j) = 2/4 = 0.5$.

Note that $nns(i,j)$ is not necessary equals to $nns(j,i)$. After defining the individual node similarity, the *overall node similarity* between two OEM records can be defined based on it. The *overall node similarity* $ns(A,B)$ for two nodes A and B is defined as follows:

$$ns(A,B) = \sum_{i \in A, j \in B} max(nns(i,j)/(\text{number of leaf nodes in } A)$$

The denominator is used as an normalization factor to ensure that the similarity lies between 0 and 1. Hence, if there exist two OEM records which are highly correlated, their corresponding node similarity should be high, or vice versa.

3.2.3 Edge Label Similarity

The computation of edge label similarity is based on the edge labels of each branch in an OEM record. Here, a branch is defined as all the edges from root to leaf. The following is the definition of edge label similarity over two individual branches:

Edge label similarity between two branches i, j $nes(i,j)$ = (number of edge labels in i that match with that in j)/(number of edge labels in i)

For example, if there are two branches i, j which are of the forms:

$$i : \longrightarrow^{a} \longrightarrow^{b} \longrightarrow^{c} \longrightarrow^{d} e$$

$$j : \longrightarrow^{b} \longrightarrow^{a} \longrightarrow^{m} \longrightarrow^{n} q$$

where the characters on the arrows are the corresponding edge labels. The computation of edge label similarity considers only the content of the edge labels and ignores the order of edge labels. For example, the edge label similarity of the above example nes(i, j) is 2/4=0.5.

Once the edge label similarity between two edges is defined, it is easy to define the *overall edge label similarity* $es(A, B)$:

$$es(A,B) = \sum_{i \in A, j \in B} max[nes(i,j)]/(\text{the number of branches in } A)$$

3.2.4 Structural Similarity

In this subsection, we will define the similarity based on the structure of the OEM record. Firstly, we consider the simplest case: a tree with leave nodes only (no internal nodes). The *structural similarity* between two nodes i, j in such a tree $nns(i,j)$ is as follows:

structural similarity $nss(i,j)$ = [number of parent of i - g(i, j)] + [number of children in i - f(i, j)]/(number of children in i + number of parent in i), where

$$f(i,j) = \begin{cases} \text{difference in number of children between } i \text{ and } j, & \text{if difference} < \text{number of children in } i\text{, or} \\ \text{number of children in } i, & \text{if difference} \geq \text{number of children in } i \end{cases}$$

$$g(i,j) = \begin{cases} \text{difference in number of parents between } i \text{ and } j, & \text{if difference} < \text{number of parents in } i \text{, or} \\ \text{number of parents in } i, & \text{if difference} \geq \text{number of parents in } i \end{cases}$$

Hence, if i has one parent and three children, while j has one parent and one child, $nns(i,j)$=2/4=0.5. The *overall structural similarity* is defined as:

$$ss(A,B) = \sum_{i \in A, j \in B} max[nns(i,j)]/(\text{number of non-leave nodes in A})$$

The rationale behind the *structural similarity* is as follows: similar data should have similar "basic" structure too. What it means by "basic" is that similar records should consist of similar node structure, though not exactly the same. For instance, "Individual Profile" or "Personal Information" usually have attributes "Name", "Sex/Gender", "Age/Date of Birth", "Address", etc. Though not necessarily the same, they should have a certain degree of relation and similarity. Therefore, structural similarity should also be considered.

3.2.5 Overall Similarity

Based on the three similarity definitions defined in the previous subsections, the *similarity between two OEM records A and B* is as follows:

$$sim(A,B) = \alpha \times ns(A,B) + \beta \times es(A,B) + \gamma \times ss(A,B), \text{ where } \alpha + \beta + \gamma = 1$$

The similarity can be adjusted by tuning the three variables α, β and γ. The rationale behind the computation of similarity into three different aspects is that it is possible to handle different types of documents. *Node similarity* and *edge label similarity* are computed separately as it is possible for two nodes in different OEM records to be similar, but of totally different edge labels (heading/subheadings). Moreover, similar data should have similar "basic" structures.

Some similar records may be characterized by similar structures together with similar data content. So, the three variables should be tuned carefully based on the nature of the documents. In handling web documents, it is possible to put the heaviest weight on *node similarity* and put a lighter weight on *edge label similarity* as well as *structural similarity.*

3.2.6 Representative Selection

After the definitions of similarity being proposed, the final step is simple: compute the similarities between potential representatives and members, and select the potential representatives with the maximum average similarity with all documents in the subset. It is noted that there may be more than one representative that is of the same average similarity. If this happens, the one with higher *importance* value should be selected. If the importance values are still the same, simply take both as the representative.

It is possible for two or more document subsets that are of similar content, but in different partitions because of no hyperlinks existing between two document subsets. Hence, after the identification of *representatives*, they should be compared with their similarity. If the similarity between the *representatives* is larger than a certain threshold, the two document subsets should be merged into one subset.

It is also possible for the *representative* of a subset to contain only very little content. For this case, if the average similarity between the *representative* and its member is too low, the *representative* is ignored. Instead, the feature vector extracted from the documents in subset should be obtained as the *representative* of the subset. However, if the documents are properly-formatted, the probability for *representatives* to contain only very few information should be low.

3.3 Incremental Update

When a new web document is added, the set of hyperlinks in the document is analyzed and the number of in-links and out-links should be counted. However, on the computation of in-links, it is not necessary to check the entire set of documents in the database. Only the subset of documents pointed at by the hyperlinks contained in the document should be checked for most of the cases. Consider the following three scenarios:

Case 1: the page is a related page to a subset of documents
If the new document is related to the subset of documents, it should contain some hyperlinks pointing to some documents in the subset. Hence, for this case, there is no need to consider other subsets of documents.

Case 2: the page is pointing to many subsets of documents
For this case, it is obvious that the document is an unrelated document to any subset of documents. Hence, this document can be ignored.

Case 3: most of hyperlinks are pointing to a subset of document, but no hyperlinks from the subset to that document
It is possible for the document to be a related document to the subset. However, based on the definition of related documents in this chapter(Proposition 1, 2 and 3), this document will also be discarded.

Based on the definition, if the document has no links to other documents, it will not be assigned to any subset. Hence, there is no need to check those subsets which the document (or the set of documents) have no hyperlinks pointing to.

It is possible for a newly added document to be one of the following types: a related document to a subset, an unrelated document to any subsets or a link document to more than one subsets. All these cases will be discussed below.

3.3.1 Documents related to a subset

If the newly added document has the entire set of links pointing to the documents in a subset and there are hyperlinks from the documents in the subset to the new document, the document can be grouped into the subset. Recalled the definitions for center pages and unrelated pages, if a document is pointing to a subset of documents, it should be checked to see if it is the center page of the subset. If it follows the requirements on Proposition 1, it should be set the center document of the subset. If it is not an unrelated document to the subset, and the new document is pointed by a document in the subset, it is also regarded as the related document in the subset, as related documents are gathered by apriori-gen algorithm based on the set of center documents.

If the newly added document is a center document, the same algorithm mentioned above is applied to gather all the documents pointed by the new document into the subset. The status of some documents in the subset may change due to the addition of new documents.

3.3.2 Documents unrelated to any subset

Based on Proposition 2, the newly added document can be determined if it is an unrelated document or not. If the document is an unrelated document, the document is ignored from the subset. If the new document is unrelated to all subsets of documents it is pointing to, the document can be ignored.

3.3.3 Documents linking up two or more subsets

It is possible for two or more subsets of documents to refer to some similar topics, but classified into different subsets. Hence a document may link up two or more document subsets. For this case, just create a new subset which is the union of all subsets. The general structure for the new subset is the union of all the general structures in original subsets. All the old document subsets will be removed. Nothing in the new subset will be changed (center nodes, related nodes, relation between OEM record and the general structure). The addition of the new document can follow the steps mentioned in section 3.3.1.

3.4 Experimental Results

In this section, we will report some of our experimental results, which show the effectiveness of the proposed algorithm. Our experiments are done on two aspects: compare the proposed algorithm with k-nn classification algorithm and compare the *representatives* with *feature vectors*.

3.4.1 Compare with k-NN

In this subsection, we will compare our proposed algorithm with k-nn classification algorithm on a set of web documents. As our proposed algorithm is different from the traditional methodologies which are mainly based on *feature vectors* and compare similarities among documents, it is difficult to compare our proposed algorithm with those algorithms. Hence, we just compare our algorithm with the most basic case: k-nn classification algorithm.

The set of experimental data we use are a set of 630 web documents that gathered based on three different topics. This set of documents is named *data set 1*. Moreover, we also gather a set of 60 documents which is totally unrelated with the set of documents extracted before (*noise documents*), which is used to compare the *noise filtering capacity* of the two algorithms. We name this set of documents *data set 2*.

# Cluster center	Precision	Recall
2	0.626049	0.685381
3	0.849492	0.791174
4	0.836933	0.695488
6	0.780295	0.630929
8	0.728929	0.580907
10	0.684615	0.397534

(a)

α_1	α_2	δ	Precision	Recall
0.4	0.35	0.2	0.676885	0.63346
0.3	0.27	0.15	0.836950	0.702256
0.25	0.22	0.1	0.852088	0.723223
0.18	0.15	0.08	0.877067	0.742108
0.15	0.1	0.06	0.880128	0.750740
0.1	0.09	0.05	0.860552	0.702155

(b)

Table 3.2: Experimental results for the first set of documents for (a) k-nn and (b) our proposed algorithm.

For each document downloaded, we will extract a feature vector of dimension 20, based on the occurrence frequency of words in the document. The feature vector extracted will be used in the k-nn classification algorithm. The following subsections will show the accuracy of the classification algorithms based on *precision* and *recall* on a noise-free and a noisy environment.

Classification on a quiet environment

In this subsection, we will evaluate the two classification algorithms based on the first set of documents downloaded. For k-nn classification algorithm, we will vary the number of k (cluster centers) and compute the precision (p) and recall (r) of the classification. While for our proposed algorithm, we will vary the set of parameters α_1, α_2 and δ to yield different results in p and r. The results are summarized in table 3.2.

Based on table 3.2, we can see that our proposed algorithm out-performs k-nn classification algorithm. However, the data set used in this subsection are well-classified and have little noise. In the next subsection, we will show the effect on the presence of noise to our algorithm and k-nn classification algorithm.

# Cluster center	Precision	Recall
2	0.332956	0.452033
3	0.389399	0.496296
4	0.608427	0.626904
6	0.751473	0.650623
8	0.751423	0.650623
10	0.726384	0.490127

(a)

α_1	α_2	δ	Precision	Recall
0.4	0.35	0.2	0.676885	0.633465
0.3	0.27	0.15	0.836950	0.702256
0.25	0.22	0.1	0.852088	0.723223
0.18	0.15	0.08	0.877067	0.742108
0.15	0.1	0.06	0.880128	0.750740
0.1	0.09	0.05	0.860552	0.702155

(b)

Table 3.3: Experimental results for the first set of documents for (a) k-nn and (b) our proposed algorithm.

Classification on a noisy environment

The experiments done in this part is exactly the same as that in the previous subsection, but the data set used is the combination of data set 1 and data set 2. That means, there are some documents in the set that are totally unrelated with other documents nor any clusters. The experimental results are shown in 3.3.

From table 3.3, it can be seen that there is no effect on the results generated by our proposed algorithm. That means, the presence of noise does not affect the performance of our proposed algorithm. However, when compared with k-nn classification algorithm, it is adversely affected and the clusters formed are not as good as that formed in the previous subsection's result.

3.4.2 Representative vs Feature Vector

The most important features in a subset are contained in the feature vector. Can *representatives* have the same property that can represent the subset of documents it belongs to? The experimental results presented in this subsection are to show that the *representative* can represent the set of documents it belongs to.

Clu ID	# docs	FV_{100}	FV_{50}	FV_{20}	FV_{10}
1	24	0.39	0.32	0.35	0.30
2	65	0.62	0.68	0.65	0.50
3	223	0.47	0.50	0.65	0.60
4	3	1.00	1.00	1.00	1.00
5	3	1.00	1.00	1.00	1.00
6	2	1.00	1.00	1.00	1.00
7	6	0.38	0.48	0.65	0.60
8	305	0.48	0.52	0.55	0.70

Table 3.4: Experimental results on the comparison of *representatives* with feature vectors.

Representative(s) is/are identified from each subset of documents based on data set 1, with α_1, α_2 and δ being set 0.18, 0.15 and 0.08 respectively. The *representatives* are compared with the feature vectors (dimension 100, 50, 20 and 10) of its subset to see how many features in the feature vectors are contained in the *representatives.*

For the document set 1, according to our proposed algorithm, the set of documents are classified into 8 subsets. They are numbered with number 1 to 8 respectively. Table 3.4 shows the results on our experiment.

Table 3.4, it can be seen that *representatives* in each document subset can represent the document subset. It is impossible for the representatives to contain all the keywords in its document subset. However, by ignoring those subsets with only few documents and considering those with large number of documents, it can be seen that the representatives usually contain more than 50% of the keywords in its subset. The representative for the cluster with ID 1 has a low percentage on the feature contained because there is a document with high *importance* and *reference*, but have few content as mentioned in section 5.

Chapter 4

Web Document Integration

In Chapter 3, we discussed the algorithm on how to partition web documents into smaller and meaningful subsets. However, the structures of the web documents in the same subset are still of large derivation. This also prohibits further operations like efficient searching of web documents. Thus, we try to integrate the subset of web documents under a standardized data structure for each subset of documents.

In this chapter, an algorithm is proposed to generate a "general structure" based on OEM data in each subset. The general structure generated can be used to store the data retrieved from the subset of documents. In the previous chapter, an intermediate step for representative selection is the transformation of web documents into OEM. The set of data will be used again in this section and based on that in the same subset, the general structure can be generated.

4.1 Structure Borrowing

In the previous chapter, a similarity definition between two web documents was proposed. For the computation of representatives discussed in the previous chapter, node similarity may contribute more under the computation of similarity between two documents. However, in this chapter, the node similarity will be ignored and the main focus will be put on the edge label similarity as well as structural similarity for the computation of the general structure.

In this section, two parameters, δ and γ, have to be pre-inputted for the identification of seeds for "general structure" generation. A set of seeds is identified from each subset of documents which their OEM data structure will be used as the base of the general structure. The criterion as the seed of a document subset is that its structure as well as its edge label is close to many other documents in its subset, so that most of the documents in its subset can employ its structure for data manipulation. Hence, a definition is given based on the requirement above:

Definition 4.1 (Potential seed for a subset) *If the document d_i in a subset of documents has edge similarity $es(d_i, d_j) > \delta$ as well as structural similarity $ss(d_i, d_j) > \gamma$ towards a document d_j in the same subset, d_i is considered as a* ***potential seed*** *of the subset. The document d_j is said to be structurally similar to (sst) d_i.*

It is obvious for us to consider the structural similarity in the identification of potential seeds. However, the edge label similarity is also considered because it is possible for two documents to have similar structures in terms of their OEM record, but of totally different content (edge label). Recalled the definition of structural similarity, it considers only the basic structure of nodes based on the number of children and parents. Moreover, not only the basic structure of a potential seed is considered, but the edge labels too will be used in the generation of the general structure. Hence, it is essential for considering the edge label similarity between documents.

However, it is possible for more than one document that have similar structure with a document in the subset. Redundant potential seeds will increase computation time for further operation and result in generating unnecessary branches in the general structure. Therefore, the minimal set of potential seeds should be extracted such that the set should have a high similarity (the edge label as well as the structural one) to as many documents in the subset as possible.

Figure 4.1 is the algorithm for identification of the set of seeds in a subset. By following the algorithm, sets of seeds can be retrieved from subsets of documents. The set of seeds should be the minimal set of documents that are structurally similar to the set of documents in the subset. After the retrieval of seeds, the general structure can be generated based on the set of seeds.

4.2 Integration of Seeds

In the previous section, all the web documents are transformed into OEM before computing their similarities. The corresponding structures are used again in this subsection. The set of seeds retrieved should be of similar structure with its documents. Hence, if all the OEM records corresponding to the set of seed documents are integrated into one large OEM record, its structure should be capable of representing all the OEM records in the subset.

It is not difficult to combine all the seed OEM records in one large OEM record. However, the large OEM record is not the minimal one as it is possible for more than one seed record to contain the same (or similar) edge. Thus, an algorithm should be employed to eliminate redundant edges. Before the algorithm is presented, a definition for redundant edge is given here.

Definition 4.2 (Redundant Edge) *If an edge in an OEM record (from root to leaf) has the same number of internal nodes and set of edge labels as another edge in the same record, and the two edges have no common nodes except the root, the pair of edges is said to be a redundant edge pair. Only one edge in the redundant edge pair is necessary and the remaining can be pruned away.*

Figure 4.1: The algorithm for identification of potential seeds.

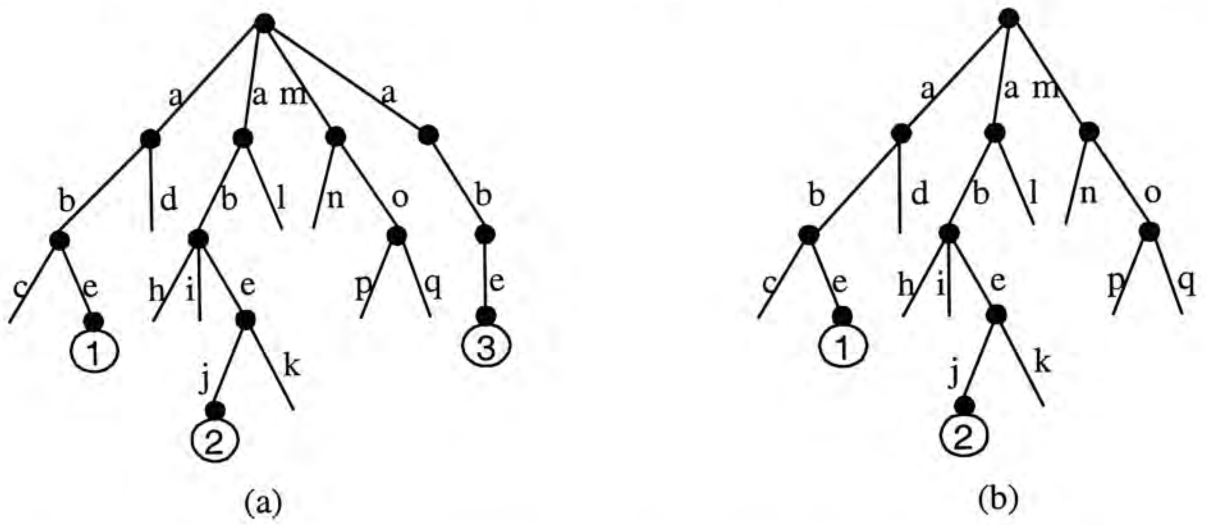


Figure 4.2: An OEM record with a redundant edge (a) and the same OEM record with the redundant edge removed (b).

Figure 4.2 shows a redundant edge pair in an OEM record. In figure 4.2(a), the edges labeled with [a]->[b]->[e] (edge with leaf node 1 and 3) are redundant edge pair. Hence, one edge can be removed from the record (the edge with leaf node 3 is removed in figure 4.2(b). However, the edge with leaf node 2 should not be removed, as there is more internal nodes than that with leaf node 1. The edge [a]->[b]->[e]->[j]->(3) has more information than the edge [a]->[b]->[e]->(1). In fact, if data can be stored in internal nodes, the edge with leaf node 1 can also be removed, but in this thesis, the model with no information stored in internal nodes is used.

The algorithm for the generation of general structure is listed in figure 4.3. Based on the algorithm, the general structures for each subset of documents are returned. Notice in the algorithm that, there is a parameter *param* for the comparison of similarity between two sets of edge labels. This is a user-defined parameter which is used to classify the similar edges from dis-similar ones. It is not necessary for all the edge labels of a redundant edge to be exactly the same as other edges. However, it should not be set too low. Otherwise, it will eliminate edges that should be kept.

```
algorithm general_stru_gen
    for each subset of documents $s_i$
        $d_i$ = the set of seeds in $s_i$
        $gs_i$ = {}      // $gs_i$: the general structure for subset i
        for each OEM record $o_j$ of corresponding document in $d_i$
            $gs_i = gs_i + o_j$
        end for

        // the basis of the  general structure is stored in $gs_i$

        for each edge $e_k$ in $gs_i$
            for each edge $e_l$ != $e_k$ in $gs_i$
                if count(edge labels in $e_k$)==count(edge label in $e_l$) and
same edge label in $e_k$ and $e_l$ > param
                    $gs_i = gs_i - e_l$
                end if
            end for
        end for

        // unnecessary edges are eliminated

    end for

end general_stru_gen
```

Figure 4.3: The algorithm for general structure generation

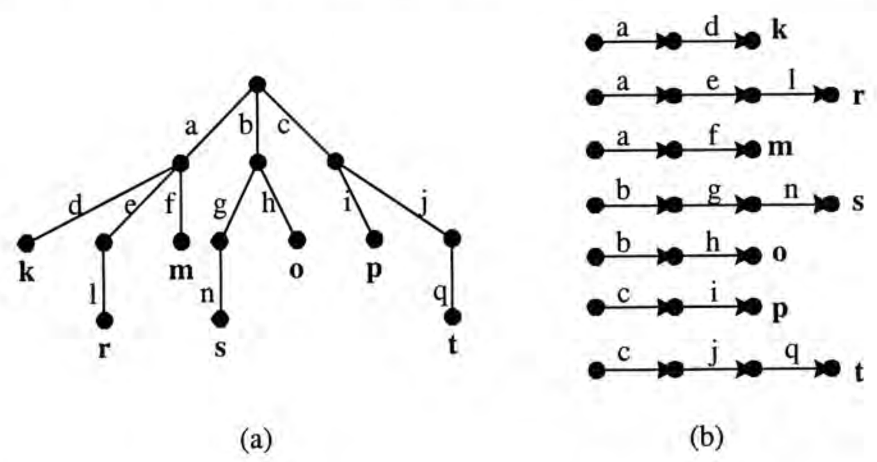


Figure 4.4: An OEM record (a) and a decomposition of the OEM record into a set of trees that all have one parent and one child in internal nodes (b).

In this chapter, the structural similarity is ignored in the computation of the general structure as all the edges in the OEM tree record are broken down into many edges with no branching in internal nodes. This phenomenon can be explained by the example in figure 4.4. As shown in the figure, the OEM record is decomposed into a set of OEM records with one parent and one child for all the internal nodes and one child for root node. The purpose for the decomposition is to ensure that the generation of general structure is not affected by the underlying structure of the OEM record. It is possible that some of the branch records are actually not useful in representing the structure. In order to extract the useful edges which can cover as many OEM records as possible, seeds are decomposed into set of OEM records following that in figure 4.4.

Then the following step is simple. For each node in the OEM records of a subset of documents, the edge in the general structure which is closest to its original edge should be selected. Though simple, there are some points that should be noticed. Recalling the algorithm for the selection of seed records, only some records in the subset of documents are selected. It is possible that there exist some edges in the subset that is not similar to any edges. If such edges are mapped into the general structure, it is possible that an error will be occurred. Moreover, it is also possible for more than one edge in the general structure that have the same similarity with an edge. Those problems will be discussed below.

Before the transformation from its original structure to the general OEM structure, a parameter λ is set as the threshold that the structure to be "borrowed". As mentioned before, it is possible for some edges that there is no edge in the general structure suitable for "structural borrowing". If there is no edge in the general structure that has similarity to the edge that has the similarity greater than λ, the edge should not be transformed as there is no edge suitable for transformation. Under this situation, three solutions are proposed:

1. eliminate the edge, as if there is no similar edge in the general structure, the edge should not be an important edge. This may result in loss of information.
2. add the edge on the general structure, but this may result in addition of unnecessary edges on the general structure and lead to an increase in its size
3. compare the node similarity with nodes of other records. For the set of nodes similar with the node, check which edge the node is transformed under the general structure. Then transform the node under that edge on the general structure. This can be sure the node abolished is of no importance on the subset of data as well as keeping the general structure pack, but the complexity of computation will be large.

The similarity mentioned above is mainly based on the definition of edge label similarity. If the OEM record has only one child and one parent in its internal nodes, the similarity used in this part will be just the same as that of edge label similarity.

The next problem is the edge selection when same similarity is encountered. If the similarity for an edge in an OEM record is the same for more than one

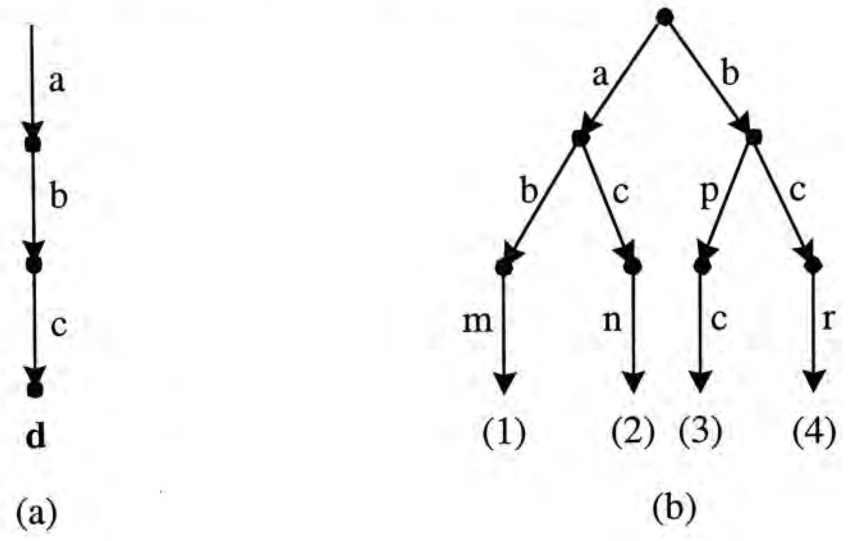


Figure 4.5: An OEM edge (a) and a general structure for a subset (b).

edge in the general structure, the edge with the same edge label in lower parts of the tree should be selected. The following figure is an example.

In figure 4.5, (a) is an edge in an OEM record. When compared with the set of edges in the general structure as shown in (b), edges labeled with (1), (2), (3) and (4) are of the similarity 0.67. They all have two out of three edge labels that match with the edge from the OEM record. However, in this case, edge (3) will be selected as the edge to store the leaf node data **d**. For an OEM data record, data is represented in a hierarchical way. The same argument also applies on the arrangement of edge labels in an edge. Edge labels should be arranged in the sense of hierarchy, from general to specific. Hence, in the example shown above, edge (3) or (4) should be selected instead of (1) or (2). However, in (4), the edge label c is in an upper level than that in (3). So, edge (3) will be selected as the edge to store the OEM data represented in (a).

4.3 Incremental Update

When a document is added into a subset of documents, the OEM record of the document should be integrated into the general structure. This can be divided into two cases: the new document is a potential seed, and the new document is a normal record. The two cases will be discussed in the following subsections.

4.3.1 New OEM record is a normal record

If the record is a normal OEM record, it should be transformed into the general structure as other member document does. There is no change on the general structure generated. That means, the new OEM record will be integrated into the general structure as other records in the subsets do. For the set of edges that cannot be transformed into the general structure, they should be treated following the solution that the transformation algorithm handles the redundant edges.

However, as the set of redundant edges accumulates, some redundant edges may gain sufficient support to be transformed into the general structure. Here, we will introduce another data structure for the storage of the set of redundant edges in each subset. The data structure will keep all the redundant edges. Under a certain period, or if the size of the set of redundant edges is increased to a certain level, the similarities between the edges will be computed. If there is a redundant edges that have a large edge label similarity with a certain amount of redundant edges in the subset (e.g. more than 1% out of the entire set of redundant edges or more than 5 edges, taking the larger one), the edge will be promoted as an additional edge in the general structure. All the edges in the set of redundant edges that is of high edge label similarity with the newly added edge will be removed from the set of redundant edge and stored under the set of general structure.

In the computation of similarity for the redundant edges, only the edge label similarity will be considered. After the transformation, the set of redundant edges are a set of OEM data with only one child for each non-leaf nodes. Thus, it is meaningless to compute the structural similarity, as that in the previous section does.

4.3.2 New record is a potential seed

If the OEM record is a potential seed based on Definition 2, the general structure should be changed by adding new edges in it. The newly added OEM record should be broken down into set of edges with internal node having one child and one parent. The similarity of each edge between the edges in the general structure is computed. By following the algorithm in chapter 4, redundant edges are eliminated and the general structure is updated.

Chapter 5

Dictionary

In this chapter, we will introduce a new data structure called "dictionary". In the previous chapters, we discussed how to classify and integrate the set of web documents based on the hyperlinks embedded in the documents as well as their structures and content. However, the issue of semantics about the keywords is still not addressed. Some related keywords that appear together frequently, can be used to identify the semantics of a document. For example, the keyword "Java" is about an object-oriented programming language or a kind of coffee. However, it is difficult to understand the relation of words without an additional data structure for the semantics between words does not exist. In this chapter, we will try to introduce a new data structure called *dictionary* to keep track of the relation among words based on their co-occurrences.

The dictionary will serve several purposes. The first purpose is, just as mentioned before, to identify the relations among words. In our proposed data structure, the dictionary keeps the information of a word, which includes the occurrence of the word, the occurrence in different subset, the position of the word located and the co-occurrence with different words. Based on these information, we can identify correlated words. For example, the keyword "Java" should have a high occurrence on subsets of documents concerning programming languages or coffee. Therefore, we expect to have a high co-occurrence between the keyword "Java" and the keyword "programming" or "coffee". Moreover, based on the occurrence and the distribution of keyword, we can also identify some "useless words" like "is", "a" or "the". They cannot help identifying document semantics, as they appear in many documents. We can prune out these words and hence can save the storage space.

Moreover, the dictionary can help representing the subset. The *representative* may not be possible to include all the keywords in its subset of documents. If solely the representative is used for representing the subset, some keywords may be missed. The representative is expected to be the one which is the most similar to all documents in the subset. However, in most cases, it is impossible for the representative to contain all the keywords in its subset. Thus, the dictionary can help representing the set of documents. In the proposed structure of dictionary, the occurrence of a specific keyword in a subset is kept. Based on this entry, we can retrieve the set of keywords for each subset quickly. In addition, by submitting a keyword, the sets of related keywords and subsets containing the keyword can be retrieved quickly.

In the following sections, we will discuss the details about the dictionary.

5.1 Structure of a Dictionary Entry

There are several important information that should be kept in the dictionary. Based on the two functionalities that mentioned in the previous section, we should expect the dictionary to keep the information below:

1. the total occurrence of the keyword

2. the occurrence of the keyword in a subset

d_i :	*word*	
	occurrence	(o_{11}, o_{12})
	occurrence pos	p_1
		...
	occurrence	(o_{21}, o_{22})
	occurrence pos	p_{15}
		...

Figure 5.1: The structure of an entry in a dictionary

3. the co-occurrence of the keyword with other keywords

Figure 5.1 shows the structure of a word stored in the dictionary, where the explanations of the entries are as follows:

d_i: the index of a word in the dictionary

word: the actual content of the word

o_{j1}: the total number of occurrence for the word in different records of subset j

o_{j2}: the number of web documents with the word appeared in subset j

p_k : the document that contains the word

Each document will be scanned and the words in the document will be added to the corresponding entry in the dictionary. If the entry does not exist, it will be added to the dictionary entry. Otherwise, the count on the corresponding word will be updated.

In the proposed data structure, we keep the occurrence information of a keyword *word* in a subset as well as the total occurrence. We also keep the occurrence position of *word* so that the relevant document can be retrieved quickly if the keyword is supplied for searching. This also helps on multiple keyword searching, which the subsets containing each keyword are extracted and the intersection of these subsets is taken as the resultant document set. Since it is possible for more than a subset of documents that contain the keyword, multiple entries for o_{i1} and o_{i2} together with its corresponding occurrence position pi will be resulted.

However, there is no co-occurrence information maintained in the dictionary. It is impossible, and unnecessary, to keep all the co-occurrence information as there are thousands of words that are co-occurred with a word in a document. It is meaningless to keep all these information. In the next section, we will discuss on how to discover the co-occurrence information based on the information maintained in the dictionary data structure.

5.2 Dictionary: Relation Identifier

The dictionary keeps no information about the co-occurrence of words. However, we can obtain this information based on the occurrence of the words in a subset. Here, we make an assumption that two words are correlated if the two words co-occur in a high frequency. In order to save the computation power, we first consider the occurrence within a subset. If there exists more than one word in the subset that have a high occurrence, the words will be selected for further examination. What it means by "high occurrence" is that if the occurrence of a word within a subset is higher than a threshold, say 50% of the word's total occurrence.

Then the set of selected words will be examined for the co-occurrence of words. The co-occurrence should be estimated under the node level, that is, if the two keywords appear in the same node, they are considered as co-occurred. The co-occurrence of the two keywords is counted. If the co-occurrence counts for both keywords are larger than a proportion of the total occurrence of the corresponding keyword, the two words are said to be related. Here, we introduce a variable ρ such that for two keywords a and b, if the co-occurrence for a keyword a with b is larger than ρ times the total occurrence of a, a is said to be related with b. If b is also related with a, the two keywords a and b is said to be correlated.

Moreover, the dictionary can be used to identify unimportant words from keywords. A keyword should be specific to a subset of documents in the sense that it should not appear frequently in many other subsets. In other words, if a word appears in many subsets with high frequency, the word should not be a keyword. Those words should be indexed with special index to indicate its unimportance so that the set of unimportant words can be ignored while performing related word searching and keyword searching.

The set of unimportant words, like "is", "a" and "the", help nothing on structural recognition nor representing the document. However, these words appear all around the documents. If a word appears in all subset of documents and if the occurrence of the word in all the subsets is larger than (total occurrence of the word) / (total number of subsets $\times$ 2) (that means, if the word appears in all the subsets with the occurrence not less than 50% of its average occurrence), the word is classified as unimportant word.

5.3 Dictionary: Complement of Representative

Though the representative can be used to represent the subset of documents, it is not probably that the representative contains all the words in the subset of documents. Hence, the dictionary can be used as the complement of representative upon searching. As the dictionary keeps the occurrence of all the words in each subset, the set of words with high occurrence will be picked out first and this serves as the complement of the representative. If a keyword is supplied as a query for web document, instead of searching from the representative, the dictionary is searched and the corresponding documents containing the keyword are returned as the answer.

Moreover, this can help speedup the searching. This is time consuming to retrieve all the representatives for the corresponding keyword. However, with the aid of the dictionary, we can retrieve the desired set of documents based on the dictionary quickly. Entries in the dictionary are arranged following the hierarchy predefined (like ordered following the alphabetical order of the keyword) and hence the searching of a keyword in the dictionary is faster than that in the set of representatives.

In addition, the dictionary can be applied to solve the problem of word ambiguity. As mentioned before, "Java" can either be a kind of coffee or an object-oriented programming language. Upon the retrieval, if there exists more than one subset which contain the query word, its correlated words will also be retrieved and returned as the preliminary query result. Users can retrieve the desired subset of documents and ignore those they do not want based on the correlated words retrieved.

5.4 Incremental Update

Upon the addition of new documents, the dictionary has to be examined and updated. Since the dictionary keeps the occurrence information about the set of documents, the incremental maintenance is simple. The dictionary will be checked to see if the entry exists or not. If the entry exists, the corresponding count and the new document should be added into the entry. However, if the entry does not exist, the entry is added in the dictionary.

Moreover, it is not necessary to perform the related word checking too fre-

Subset ID	# Document	Representative URL
1	351	http://www.cse.cuhk.edu.hk/pgm/ug.html
2	4	http://www.cse.cuhk.edu.hk/privacy/statement.html
3	4	http://www.cse.cuhk.edu.hk/internet/internet.html
4	1	http://www.cuhk.edu.hk/eco/index.html
5	8	http://ihome.cuhk.edu.hk/ z044086/index.html
6	324	http://www.acm.org/sigmod/record/credits.html
7	365	http://www.acm.org/sigmod/record/index.html
8	438	http://www.amd.com/about/investor/1998annual/annual.html

Table 5.1: The classification on the set of documents.

quently. The checking can be done on batch or upon request. The incremental update of the dictionary is simpler when compared with other algorithms.

5.5 Experimental Result

In this section, we report some experimental results based on the searching by representatives and the dictionary. 2,000 web documents are gathered based on three main topics: 1) The Computer Science and Engineering Department of the Chinese University of Hong Kong [30]; 2) ACM Homepage [31] and 3) AMD Homepage [32]. Then based on the UNIX command "*wget*", a set of web documents are gathered. Then based on the algorithm discussed on Chapter 3, the set of documents are partitioned as shown in table 5.1.

The following subsections will show the searching results based on the search of keywords, ambiguous words and related words.

5.5.1 Search based on keyword

In this part of the experiment, we try to search for the documents based on the submitted keywords by using representatives. The keywords selected are CUHK, SIGMOD and AMD. Table 5.2 shows the results of the searching. The second column is the subset IDs that contain the keyword. The third and the

Search word	Subset	Dic retrieved for related subset?	Dic retrieved for unrelated subset?	Keyword contained in unrelated subset?
CUHK	1,2,3,4	No	Yes	No
SIGMOD	6,7	No	Yes	No
AMD	6,8	No	Yes	Yes

Table 5.2: The result based on keyword search.

Search word	1R	1S	2R	2S	3R	3S	4R	4S	5R	5S	6R	6S	7R	7S	8R	8S
Database	N	Y	N	N	N	N	N	N	N	N	Y	Y	N	N	N	Y
Conference	N	Y	N	N	N	N	N	N	N	N	N	Y	N	Y	N	Y
Computer	Y	Y	Y	Y	Y	Y	N	N	N	N	N	Y	N	Y	N	Y

Table 5.3: Search results when submitting ambiguous words.

fourth column of the table indicate if the dictionary is retrieved while checking the existence of the keyword in the related subsets and unrelated subsets respectively. The last column indicates if the keyword is contained in the unrelated subsets. The result shows that if the keyword submitted is related to the document set, the representative will usually contain the keyword. Note that the keyword "AMD" also appears in the subset 6 (documents from SIGMOD pages) but have to be retrieved through dictionary.

5.5.2 Search by submitting ambiguous words

In this subsection, we will try to submit some ambiguous words to check if the corresponding representative contains the words or not. There are 8 subsets returned after the classification of documents. The table below shows whether the representative (R) and the subset (S) contain the query word or not. The below is the result:

It shows that if the word submitted for query is ambiguous, the representative may not contain the word and the dictionary is required for the search of the

home (185)	computer (523)	department (176)	hong (240)
science (240)	research (269)	engineering (299)	kong (205)
chinese (308)	please (145)	university (296)	system (200)
cuhk (283)	software (151)	information (185)	2000 (296)
systems (174)	network (251)	new (152)	

Table 5.4: The related words returned for the keyword "CUHK".

acm (2354)	management (2887)	library (1442)	systems (2676)
digital (1472)	conference (1449)	database (2805)	data (2045)
information (1714)	international (1306)		

Table 5.5: The related words returned for the keyword "ACM".

word. Thus, for searching of ambiguous words, the dictionary should be used.

5.5.3 Retrieval of related words

In this part of the experiments, we try to search for some keywords from the document set and see which "related words" will be returned. Here, we try to search for two keywords: CUHK and ACM. The search results are shown in the following tables. Words in the tables are the corresponding related words retrieved. Digit in the blanket following the word is the co-occurrence of the corresponding word with the keyword inputted.

According to the search result, the related words are retrieved successfully: "Computer" and "Science" can be retrieved from input of CUHK (one of the seeds for gathering web documents is the Computer Science and Engineering Department of CUHK) and "International" and "Conference" for ACM (the ACM homepage is another seed). However, since the number of subsets is too small, the set of keywords generated are not as clear as expected. For example, the word "please" is considered as a related word with "CUHK". If more documents are inputted, a better result may be obtained.

Chapter 6

Structured Data Manipulation: IR-Tree

From this chapter onwards, we will discuss an integration of indexing and clustering techniques to improve the performance of indexing. In the next chapter, we will discuss the new indexing algorithm under Metric space; and in this chapter, we will discuss the integration of the two techniques under Vector space.

6.1 Range Search vs Nearest Neighbor Search

Range search and nearest neighbor search are the two most common operations for an index structure upon searching. Hence, while comparing different indexing structures, the performance for the two query operations is usually compared. However, there are not many studies about the performances of the range search with knn-search. R-Tree and R*-Tree are non-deterministic indexing structures in the sense that the sequence of data insertion will affect the arrangement of data and the tree structure. Therefore, the performance of knn-search will be affected as traditional knn-search algorithm is done on a depth-first manner. In this section, we will compare the performance of range search with that of

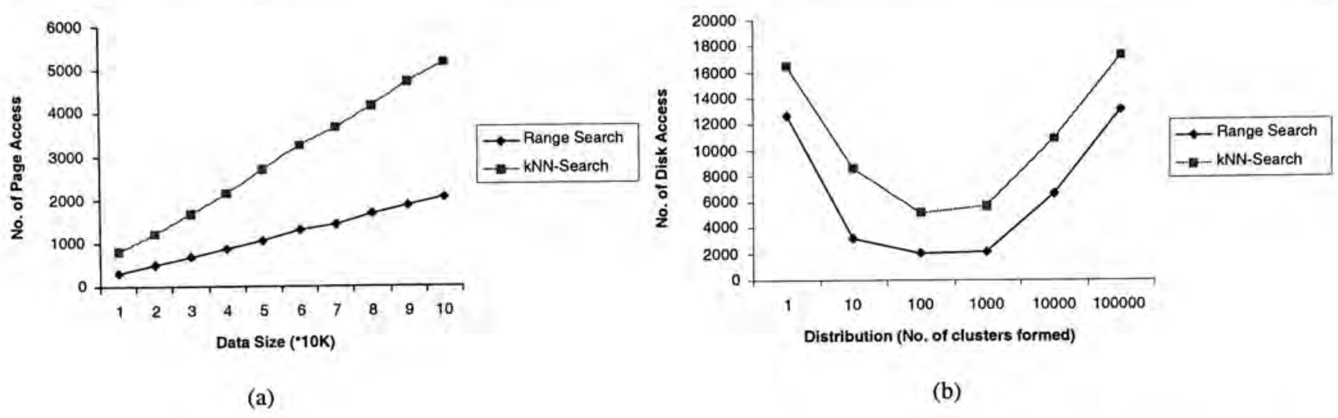


Figure 6.1: The performance of range search and knn-search on varying (a) data set size and (b) distribution of data.

knn-search under different aspects.

Figure 6.1 shows the performances of range search and knn-search on varying the dataset size and the distribution of the data respectively. The data plotted in the figure is the average of 100 queries. For all queries, the closest 100 records from the query points have to be retrieved. The distance between query point and the corresponding 100-th closest point is computed and used as the radius for the range search. 50% of data is clustered following the Incremental-DBSCAN clustering algorithm [22] with the parameter setting listed in table 6.1 for the data set used in figure 6.1 (a). For figure 6.1 (b), the parameter setting is listed in table 6.2.

As shown in figure 6.1 (a), the performance of the range search is better than that of the knn-search under a clustered environment, which reduces the number of page accesses by around 50%. The radius specified for range search limits which nodes should be visited. However, there is no constraint placed on which node to visit until the number of retrieved data points reaches k. Figure 6.1 (b) shows the performances of the two search operations on different data distributions. The performance of range search is poor when there is no cluster identified and all the data are in one cluster. There are many studies that examine the performance of R-Tree [38, 5]. The results show that if the data is uniformly distributed, only some active partitions will be involved in the split and all remaining dimensions are kept unchanged. The overlapping of bounding rectangles is large and the performance of range search is deteriorated (the curse of dimensionality). However, the performance of knn-search is still worse than that of the range search.

In this chapter, we will propose an algorithm that can transform a knn-search into a range search, which is less expensive. We will discuss the transformation algorithm and compare the performance of the traditional knn-search algorithm and the transformed range search algorithm in later sections.

6.2 Why R*-Tree and Incremental-DBSCAN?

In chapter 2, various vector-space indexing methodologies as well as clustering algorithms are introduced. In this chapter, I will discuss why R*-Tree and Incremental-DBSCAN are selected as the indexing and clustering algorithm used in IR-Tree.

From previous studies [34], there are pros and cons for employing bounding rectangles as well as spheres for indexing structures. For rectangle, it has better space management and hence less dead space, or overlapping, is resulted. Therefore, it is more efficient for performing range-search for indexing structure using hyper-rectangles like R-Tree. For bounding spheres there is an important property: for any point p bounded in the bounding sphere, the distance between p and the center of bounding sphere must be smaller than the radius of that sphere. Thus, indexing structures by using bounding spheres (SS-Tree and SR-Tree) have a better performance for knn-search, even though there are more overlapping.

In this thesis, instead of combining both spheres and rectangles in an indexing structure (like SR-tree does), we will try to integrate the Incremental-DBSCAN into R*-tree structure. The rationale behind selecting Incremental-DBSCAN is that it is a sphere-based clustering technique. Moreover, it is an incremental clustering algorithm, which is not for most of existing clustering algorithms. Finally, as the R*-tree can also help in searching for directly density-reachable objects for Incremental-DBSCAN, there's no need to build another data structure to keep track of the clusters. In the later parts of this chapter, we will present how Incremental-DBSCAN is integrated into R*-tree in order to enhance its searching efficiency.

6.3 IR-Tree: The Integration of Clustering and Indexing

6.3.1 Index Structure

The structure of an IR-Tree is divided into two parts: the first part is a tree structure, which is based on R*-tree, while the second part is a data structure for maintaining cluster information, which is based on Incremental-DBSCAN. There are two different types of nodes associated with the tree data structure: the leaf node and the non-leaf node. A leaf of an IR-Tree for data with dimensionality M has the following structure:

$$L : (I, E_1, ..., E_n), m_L \leq n \leq M_L$$

$$E_i : (p_i, data)$$

In a leaf L, there are n entries $E_1, E_2, ..., E_n (m_L \leq n \leq M_L)$, where each E_i contains a point p_i and its attribute *data*. All its entries are bounded in its bounding rectangle I of dimension M, with its structure $(I_0, I_1, ..., I_{M-1})$, where

I_i is the bounding area of I on the i-th dimension. The structure of a leaf node in IR-Tree is the same as that in R-Tree or R*-Tree.

The structure of a non_leaf node in IR-Tree is as follows:

$$N : (I, cnt\_point\_contained, C_1, C_2, ..., C_n), m_L \leq n \leq M_L$$

$$C_i : (child\_pointer)$$

In a non-leaf node, it consists of n entries. As that in a leaf node, the bounding rectangle I bounds all its children. Each entry C_i contains a *child_pointer* pointing to its subordinate. The subordinate of a non-leaf node can either be a non-leaf node or a leaf-node. There is a variable, *cnt_point_contained*, added for all the non-leaf nodes which keeps the total number of data points contained in its subordinates. This count will be used for the knn-search algorithm in IR-Tree.

Besides the basic tree structure, we also introduce a data structure keeping the cluster information. The structure of a cluster is as follows:

$$CL : (CL_1, CL_2, ...)$$

$$CL_i : \{p_j\}, cnt(p_j), centroid, interval\_radius$$

A cluster CL_i keeps all its member entries p_j and the total count on the number of points *cnt(p_j)*. A centroid is computed for each cluster, which is the average of all the points in the cluster, and it is stored in the variable *centroid*. The variable *interval_radius* is an array which keeps the radii of the cluster on various percentage of point count from the centroid. For example, if the array *interval_radius* is of dimension 10, $interval\_radius_0$ is the radius from centroid the radius specified, 10% of the points in the cluster are bounded. More detail about *centroid* and *interval_radius* will be discussed in the later subsections.

6.3.2 Insertion of IR-Tree

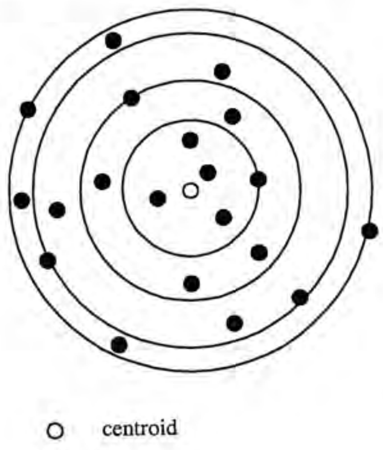
Inserting a point to an IR-tree is divided into two parts: the insertion of the point into the tree data structure, and the addition of the point to the corresponding cluster.

The insertion mechanism for the tree structure of the IR-Tree is more or less the same as that of an R*-Tree. A point is inserted in the node which contains the point, or requires the least enlargement on the bounding rectangle. If there exist two or more nodes which meet the requirement, the one with less children point will be selected. If the number of points in these nodes are the same, one of them will be selected randomly. If overflow happens, forced reinsert algorithm will be performed as mentioned in [4].

The difference on the insertion of a point between R*-Tree and IR-Tree is the update of children point count *cnt_point_contained*. If there is no split occurred on the inserted node, the variable *cnt_point_contained* are increased by 1 for the inserted leaf and all its parents. Otherwise, the total number of children points are counted again for all nodes involved in the split.

The cluster insertion algorithm for the IR-Tree is the same as that of Incremental-DBSCAN [21, 22]. For each point inserted, if there are more than *MinPts* bounded within *ddr*, a cluster is formed. The newly formed cluster may merge with other cluster(s), or exists as its own. It is also possible for the point to be a member of an existing cluster. For more details, please refer to [21, 22].

Besides the update of clusters, the information of the corresponding clusters is also updated. The *cnt*, *centroid* and the *interval_radius* of C should be updated if the cluster C is being updated. The *centeroid* for the cluster is computed by the following equation:



(a)

% quantile	radius
25 %	0.25
50 %	0.45
75 %	0.61
100 %	0.71

(b)

Figure 6.2: (a) A cluster with 20 points and (b) the table interval_radius

$$centroid_i = \{\sum_{p \in C} p_i\}/cnt(p)$$

that means, the i-th dimension of the *centroid* is the average of the i-th dimension for all the points p in the cluster C. Once the *centroid* is computed, the distance between the *centroid* and the points in C are computed and sorted. Then the array *interval_radius* can be computed based on a pre-defined variable *interval* as follows:

$$interval\_radius_j = dist(centroid, p\_sorted_{j*cnt(C)/interval})$$

which means, the set of sorted distance are partitioned into *interval* parts and the longest distance within the partition is stored in the array *interval_radius*. Figure 6.2(a) is a cluster with 20 data points and (b) is the array *interval_radius* with *interval* being set to 4.

As the update of the tree structure and the cluster structure are independent of each other, they may be done in an asynchronous manner: the update of the cluster structure can be done in batch; while the update of the tree can be done in real time. It is a time-consuming task to update the cluster information and it is possible to update the clusters in batch (or offline). However, it is also possible to update the cluster information for each addition of data point.

6.3.3 Deletion on IR-tree

The deletion of an element from the IR-tree is also divided into two parts: deletion of the entry from the cluster structure and the deletion of entry from the tree.

Deletion of an entry from the tree structure of the IR-tree is just the same as that in R*-tree. It removes the entry from the leaf node and update its and its parents' bounding rectangles. If underflow happens, those underflow entries and nodes are removed and reinserted. Similar to the insertion algorithm, the count on the number of children points *cnt_point_contained* for the affected nodes should be updated.

Deletion of an entry from a cluster is also straight-forward. For the entry p to be removed, if it belongs to a cluster, p should be removed from the cluster. If there is insufficient support for the cluster after the removal of p, a split or a removal on the cluster may be occurred. For more details about the splitting of a cluster, please refer to [22]. After the deletion of the entry, *cnt*, *centroid* and *interval_radius* should also be updated for those affected clusters.

The insertion of a point to a cluster can be delayed and batched. However, the deletion of a point from a cluster must be done before a knn-search is done. If knn-search is done on a real-time manner, the deletion of a point from a cluster should also be done in real-time. The *centroid* is unnecessary to be updated frequently, but the variables *cnt* as well as *interval_radius* must be updated to ensure the consistency of these two variables.

6.3.4 Nearest Neighbor Search

Most traditional SAMs like R-tee, R*-tree, SR-tree and SS-tree employ depth first search for knn-search. However, in many cases, most of the nodes in a tree have to be visited. Hence, knn-search usually results in poor performance. In this subsection, we will present how we can improve the performance of knn-search by applying the modifications on the tree structure and the addition of the cluster structure in the IR-Tree. By applying the cluster structure, the expensive knn-search can be transformed in a less expensive range-search, resulting in a better performance on knn-search. Moreover, based on the variable *cnt_point_contained*, knn-search can be done on a breadth first manner, so that less page access is resulted. In the following subsections, we will discuss the two modified algorithms and the conditions for applying these algorithms.

KNN-Search with Clusters: Virtual Radius

There is no range specified for a knn-search, and hence, knn-search is usually done in a depth-first manner. Nodes are not visited only if the number of data points retrieved reaches k and the accessing bounding boxes contain no data point that is closer than any retrieved points. Unnecessary node retrieval is resulted if the data closest to the query point is located in the last leaf. In this section, we will propose a concept of *virtual radius* (*VR* for short). Our approach guarantees that the k data records closest to the query point will be bounded within *VR*, such that a knn-search can be transformed into a range search based on *VR*.

The computation of *VR* is based on the clusters retrieved. For each cluster identified, there are three variables associated: *centroid*, *cnt*, and *interval_radius* upon query. The distances between the query point q and the centroid of clusters are computed and sorted in ascending order. The *VR* is computed based on two

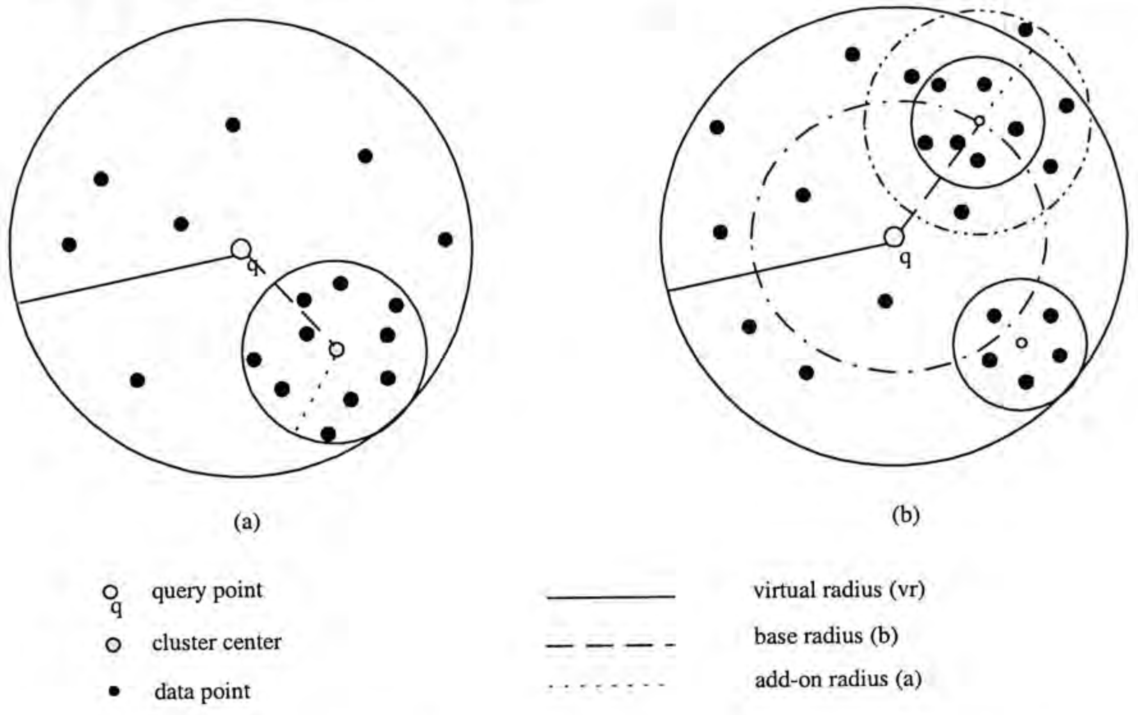


Figure 6.3: The generation of the virtual radius (VR) under a (a) single cluster and (b) multiple clusters environment

parts: *base radius* and *add-on radius*. The *base radius* is the minimal distance between the query point q and the set of centroids. For example, if the distance between q and the centroid of a cluster is 0.1, *base radius* is equal to 0.1. The *add-on radius* is the radius of the corresponding cluster that bounds more than k data points. The value of *add-on radius* can be obtained from the *interval_radius*. Figure 6.2 shows a cluster with 20 data points. From figure 6.2 (b), we can see that the radius of the cluster which bounds 50% of data points (10 data points) is 0.45. Thus, if the value of k for the knn-search is 10 and the clusters shown in figure 6.2 is examined, *add-on radius* is equal to 0.45. It is possible for VR to bound more than one cluster. If this happens, all the points bounded by VR should be counted. Figure 6.3 shows how VR is computed in a (a) single cluster and (b) multiple clusters environment.

After the computation of VR, a range search is performed by taking VR as the radius. The size of the returned set of points should be larger than the total number of points required (k). Only the k points closest to q will be returned.

KNN-Search without Clusters: Breadth-First KNN-Search

We discussed the proposed knn-search algorithm based on the clusters identified and the generation of VR. However, it is possible that the VR generated is invalid, if either one of the following two cases is true:

- the *base radius* is larger than $2\times Eps$, or
- there is insufficient cluster support or no clusters identified

What it means by "insufficient cluster support" is that the total number of points bounded by all clusters is smaller than k, the number of points required. Under these two cases, the algorithm proposed in the previous subsection fails to operate. Thus, in this subsection, we will propose another knn-search algorithm, which is on a breadth-first manner.

Before presenting the breadth-first knn-search algorithm, we will introduce two distance functions *MIN_DIST* and *MAX_DIST*. *MIN_DIST* is the minimal Euclidean distance from a point to a bounding box; while *MAX_DIST* is the maximal case. The definitions of the two distance functions between a point and a bounding box of dimension n are as follows:

$$MIN\_DIST(point, box) = \sqrt{\sum_{i=1}^{n} min(point_i, box_i)^2} \tag{6.1}$$

$$MAX\_DIST(point, box) = \sqrt{\sum_{i=1}^{n} max(point_i, box_i)^2} \tag{6.2}$$

where $min(point_i, box_i)$ and $max(point_i, box_i)$ are the minimal and maximal distance respectively between the point, *point*, and the bounding box, *box*, on the i-th dimension. The $min(point_i, box_i)$ is equal to 0, if the query point lies between the bounding box on the i-th dimension.

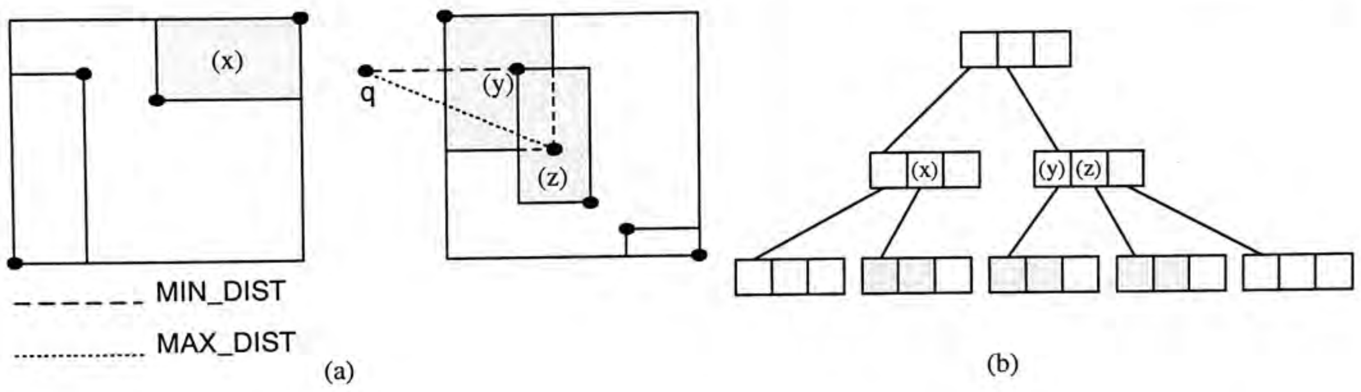


Figure 6.4: The selection of desired nodes and the pruning of unnecessary nodes upon knn-search

After the definition of the two distance functions are introdced, we can discuss the new knn-search algorithm. The algorithm is performed on the tree under a breadth-first manner from root to leaf. All the nodes in the same level that are not being pruned are sorted according to their MAX_DIST. The first n nodes, which contain the k points closest to the query point q, will be selected. The MAX_DIST of the last node is defined as $LAST\_MAX\_DIST$ and all the remaining nodes with their MIN_DIST larger than $LAST\_MAX\_DIST$ will be pruned away. Only the children of the selected nodes will be examined in the next level. The algorithm stops when leaf nodes are visited and the k closest data points are returned as the answer. Figure 6.4 is an example for the algorithm, where k is set to 3 (the three points closest to q have to be retrieved). x is the closest bounding rectangle from q. However, there are only two points bounded in x. Hence, box y, the next bounding bounding box closest to q, is selected. However, the MAX_DIST for bounding box y is longer than the MIN_DIST of bounding box z and hence the bounding box z will also be included in the set of nodes and their children will be examined. All the remaining bounding boxes in the same level are pruned away.

6.3.5 Discussion on IR-Tree

Generally speaking, the difference between the IR-Tree and R*-Tree is mainly on the algorithm for knn-search. The knn-search algorithm for the IR-Tree is divided into two different algorithms: the computation of virtual radius and transform the knn-search into a range search when one or more cluster is located close to the query point, and the breadth-first knn-searching algorithm when there is no cluster support for the query point. For the first case, it requires the support of clusters which is identified based on the Incremental-DBSCAN clustering algorithm. Clustering algorithms are usually slow in performance when compared with traditional SAMs. Thus, it is not beneficial to use the first knn-search algorithm for IR-Tree if the dataset is not intended to be clustered. However, if the dataset is intended to be clusterized, the first knn-search algorithm for IR-Tree may be applied.

Instead of being limited by the clusters identified, the breadth-first knn-search algorithm for the IR-Tree is not limited by the existence of clusters. No matter there is cluster discovered or not, the second proposed algorithm still works. In the next section, we will compare the performances of the two algorithms and have a further discussion on the result.

6.4 Experimental Results

In this section, we will present some results based on the experiments we did. We will concentrate on the number of page accesses and the CPU time required on the knn-search based on different indexing structures. The insertion and deletion algorithms are mainly based on the existing algorithms (R*-Tree and Incremental-DBSCAN). There are much research effort that compares the performance of those algorithms with other proposed algorithms. Hence, we only compare the

Parameters	Values
Dimension of data set	10
% of data being clustered	50%
Max no. of children in a node (leaf and non-leaf)	10
$MinPts$	20
Eps	0.005

Table 6.1: Parameter settings for the experiments on the general performance of R-Tree, R*-Tree and IR-Tree.

performance of our proposed algorithm with other existing algorithms based on knn-search. We will compare the performance of the knn-searches among IR-Tree, R-Tree and R*-Tree. All the experiments are done on a Sun Microsystems workstation, Sun Ultra 5/270, with 128Mbytes of main memory and with the OS of Solaris 2.6. All programs are implemented in C++. The size of nodes is set to 8192 bytes while the size of data area associated with each leaf entry is 512 bytes. All the data are synthetic and their values are ranged from -1 to 1 for all dimensions. The number of k for knn-search is set to 500 for all queries. 100 trials are done and the average is taken as the final result. Special parameters that will be tuned in the experiments, like the data set size, dimensionality of the data set and the distribution of the data, will be specified in the corresponding experiment.

6.4.1 General knn-search performance

In this subsection, we will estimate the performance of knn-search for i) R-Tree, ii) R*-Tree and iii) IR-Tree. The size of data sets is varied from 10,000 to 100,000 data points and the parameters used in various indexing data structures are listed in table 6.1.

The performance on the knn-search for the three different indexing structures is shown in figure 6.5. From the figure, we can see that IR-Tree out-performs R-

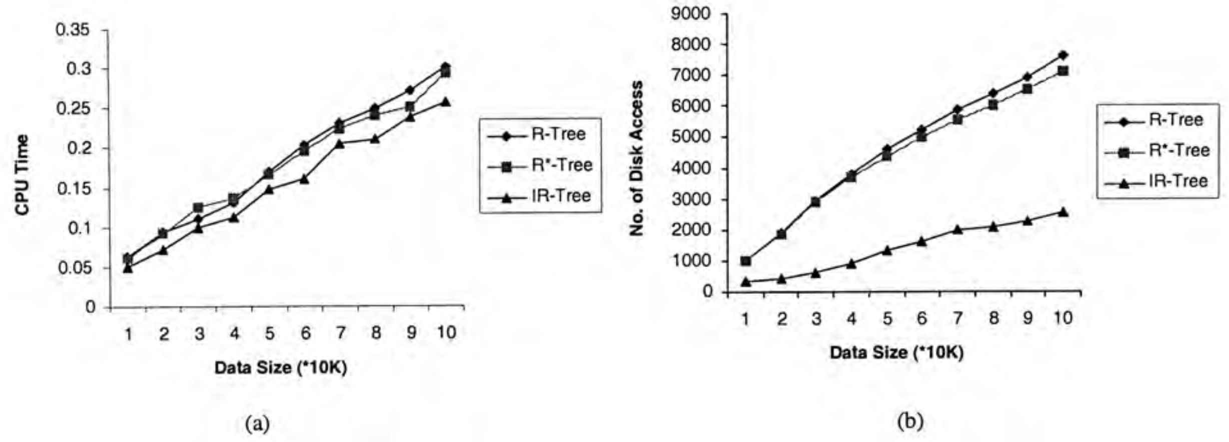


Figure 6.5: The general performance of R-Tree, R*-Tree and IR-Tree on knn-search based on (a) CPU time and (b) disk access.

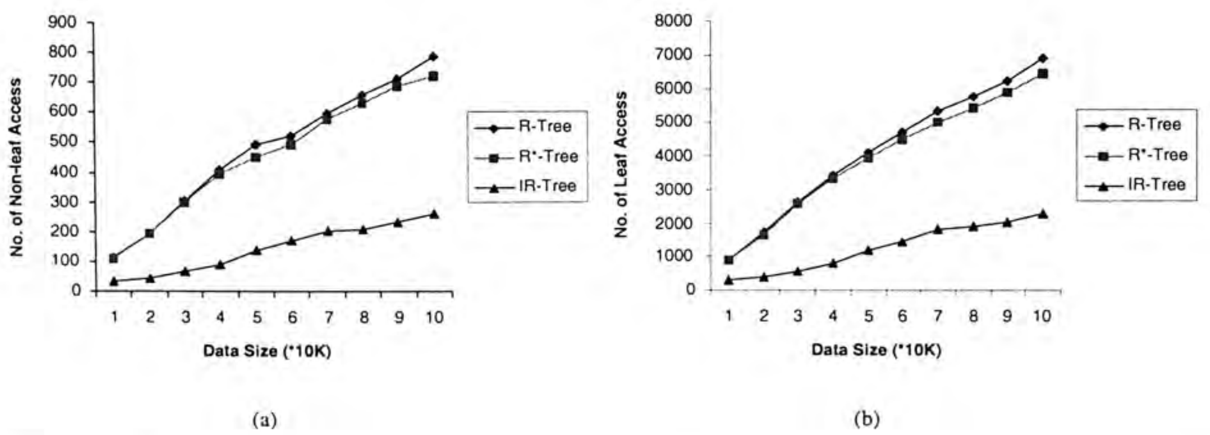


Figure 6.6: The leaf and non-leaf access for the three indexing structures.

Parameters	Values
Data size	50,000
% of data being clustered	50%
Max no. of children	10
MinPts	20
Eps	0.0002**dimension*

(a)

Parameters	Values
Data size	50,000
Dimension of data set	10
Max no. of children	10
MinPts	20
Eps	0.005

(b)

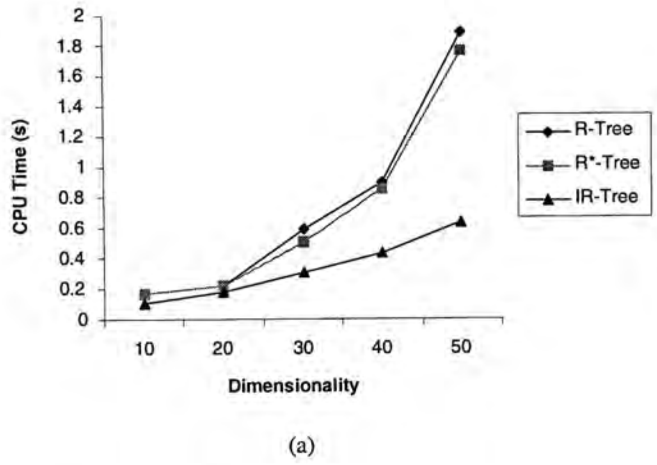
Table 6.2: Parameters for data sets varying (a) dimensionality and (b) distribution.

Tree as well as R*-Tree. As the tree insertion algorithm for IR-Tree is exactly the same as that of the R*-Tree (except the variable added for counting the number of children points), the improvement on the performance of the knn-search is from the new searching mechanism of IR-Tree. Figure 6.6 shows the number of leaf node and non-leaf node access. The ratios for leaf and non-leaf node access are similar for three different indexing structures.

6.4.2 Performance on Varying Dimensionality and Distribution

In this subsection, we will show the performance of the three indexing structures on varying the dimension of data set as well as the distribution. Table 6.2 lists the setting of parameters. Those parameters listed in table 6.2 (a) are for the data set on varying the dimensionality, while those parameters listed in table 6.2 (b) are for the data set on varying the data distribution. Noted that in table 6.2 (a), the variable *Eps* is a variable depending on the dimension of data. The increase in dimensionality means the decrease in the density of data. Hence, the *Eps* for a cluster should be adjusted according to the dimensionality.

Figures 6.7 and 6.8 present the results on the CPU time required and the number of disk accesses on varying the dimensionality and data distribution



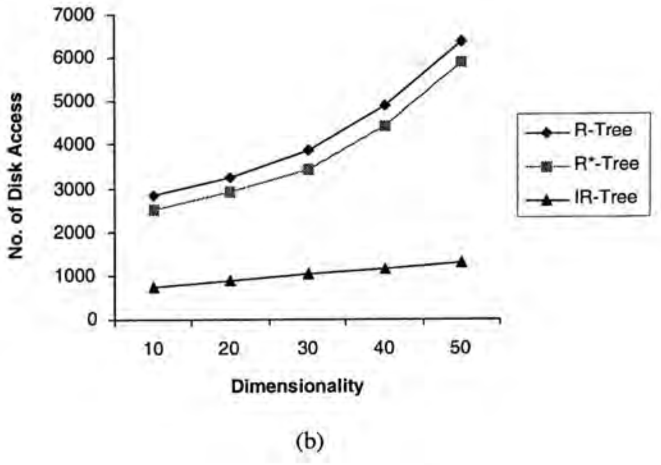
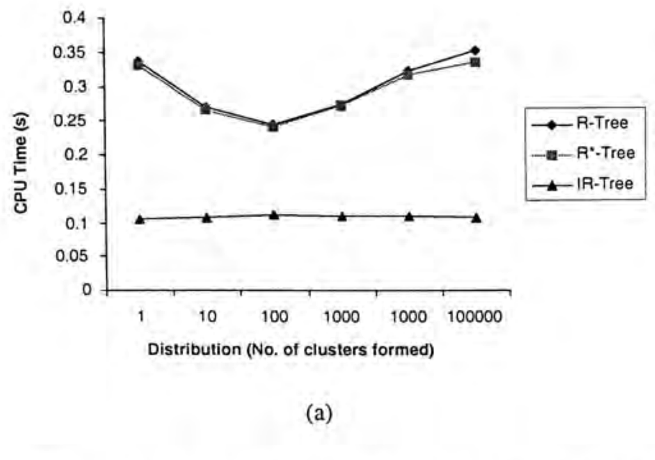


Figure 6.7: The performance of the three indexing structures on varying the dimensionality of data.



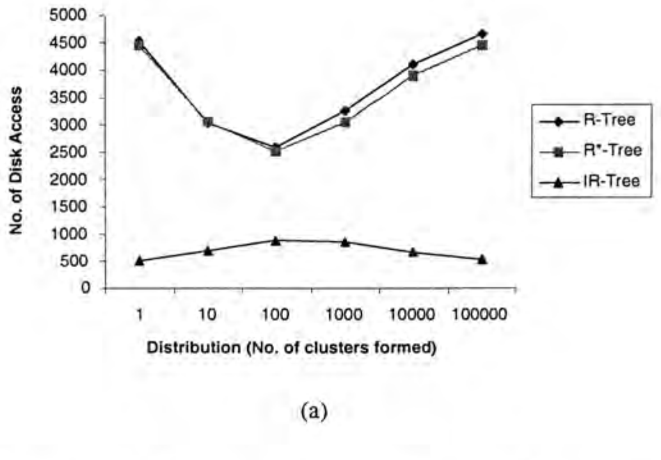


Figure 6.8: The performance of the three indexing structures on varying the data distribution.

respectively. From figure 6.7, we can see that the performance of the R-Tree and R*-Tree deteriorates in a much faster rate when compared with that of the IR-Tree. That means, the effect of dimensionality curse to IR-Tree is not as severe as that to the R-Tree or R*-Tree. In figure 6.7 (a), we can see that there is a 600% increase in the CPU time needed for the IR-Tree from dimensionality of 10 to 50. As the dimensionality increases, it requires more looping for the computation of distances between the query point and the bounding boxes/data points and hence more CPU time is required.

From figure 6.8, we can also see that the performance of knn-search for IR-Tree out-performs that for R-Tree and R*-Tree upon varying the data distribution. However, for R-Tree and R*-Tree, the performance is optimal for around 100 clusters, while that of the IR-Tree is optimal for only one cluster and no cluster identified. Figure 6.9 shows the performance of knn-searches for IR-Tree with and without using clusters (all the queries are ensured to have cluster support). From the figure, we can see that the performance of the search algorithm is better without cluster support for the data with low dimensionality. When the dimensionality increases, the performance of searching with clusters deteriorates in a slower rate than that without using clusters. When the dimensionality is low, less overlapping for bounding rectangles is resulted upon splitting. Hence, the performance of searching without using clusters, which is on a breadth-first manner, is better as less overlapping implies efficient pruning of unwanted bounding rectangles. However, when the dimensionality increases, the splitting of bounding rectangles during insertion is less clear-cut and excessive overlapping is resulted. Thus, the performance of searching without using clusters deteriorates in a faster rate than that with using clusters.

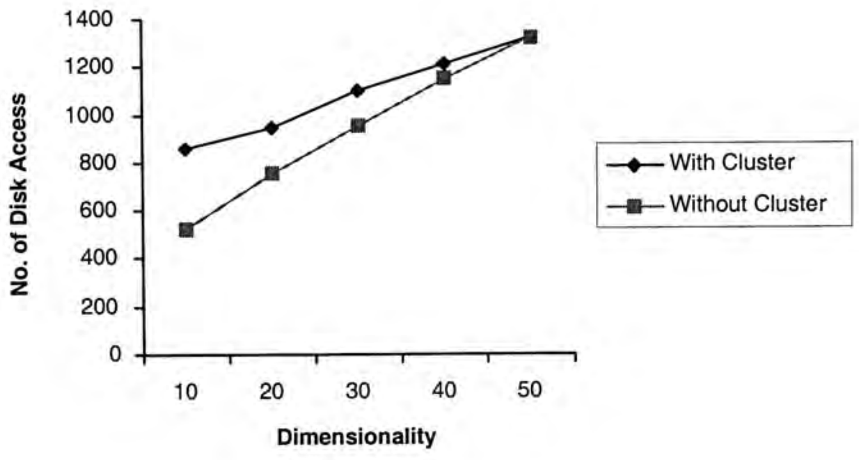


Figure 6.9: The performance of IR-Tree with and without using virtual radius on different dimensions.

Chapter 7

IM-Tree: An Review

In the previous chapter, we discussed in detail on how to apply the clustering algorithm, Incremental-DBSCAN, on vector-space indexing structure, R-Tree, to improve the performance of R-Tree and to solve some existing problems (like the problem of dimensionality curse) that the R-Tree and its derivations are facing. We already showed in the experiment section that the integration of Incremental-DBSCAN and R-Tree can improve the performance of knn-search. In this chapter, we will extend this idea on the integration of a clustering algorithm and a metric-space indexing technique. We will also have a discussion on which indexing and clustering algorithms to be used and compare their benefits and weaknesses to see if those algorithms are suitable for the integration or not.

7.1 Indexing Techniques on Metric Space

In this section, we will have a discussion on existing metric space indexing techniques. However, before discussing the indexing techniques, we will first define the metric space. Then we will introduce some existing metric space indexing techniques.

7.1.1 Definition

Formally speaking, a *metric space* is a pair, $M = (D, d)$, where D is a domain of feature values (the indexing *keys*) and d is a distance function with the following properties [16]:

1. $d(O_x, O_y) = d(O_y, O_x)(symmetry)$
2. $d(O_x, O_y) > 0$ $(O_x \neq O_y)$ and $d(O_x, O_x) = 0(nonnegativity)$
3. $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)(triangularinequality)$

Similar to vector space indexing structure, indexing a metric space means that it must provide efficient support for answering *similarity queries* [16]. The similarity is measured based on the distance function d. The indexing structures in metric space also support *range queries* and *k nearest neighbor queries*. Their definitions are given below:

Definition 7.1 (Range Query) *Given a query object $Q \in D$ and a maximum search distance $r(Q)$, the range query* ***range****$(Q,r(Q))$ selects all indexed objects O_j such that $d(O_j, Q) \leq r(Q)$.*

Definition 7.2 (k Nearest Neighbor Query) *Given a query object $Q \in D$ and an integer $k \geq 1$, the k-NN query* ***NN****(Q, k) selects the k indexed objects which have the shortest distance from Q.*

7.1.2 Metric Space Indexing Algorithms

There are already many algorithms proposed for indexing metric space data. However, for most of the proposed algorithms, the indexing tree built are static. That means, once built, the tree structure cannot be changed. In this section, we will have an introduction on these proposed algorithms.

Static Metric Space Indexing

The authors in the FastMap algorithm [23] try to transform the problem of metric space indexing to vector space indexing. The algorithm first computes the distances between the elements and then organize the set of distances in pairwise into a matrix. Then the matrix is transformed into a set of low-dimensional points such that the set of points can be indexed by a SAM.

However, the FastMap algorithm associates with a problem that it assumes the data set is static. In order to handle the dynamic dataset, the authors introduce approximation errors in the mapping process. Hence discrepancy may occur.

Another indexing algorithm, Vantage Point (VP) Tree [15], first selects a set of *vantage points* and the dataset is partitioned into corresponding partitions based on the distances between the objects and the vantage points. Usually the median of the set of distances will be used as the separator for the partitioning of objects in order to obtain a balanced split. The splitting of objects is done recursively in order to split the set of objects into smaller subsets and the nodes (partitions) are arranged in the hierarchy. The authors of the MVP-Tree [11] extend the idea by introducing multiple vantage points. They also introduce the idea of pre-computed distances in order to reduce the number of distance computation upon query.

The GNAT design [12] applies the generalized hyperplane [49] for the partitioning of the objects. For each partitioning, two reference objects will be selected from the set of objects and all the remaining objects are assigned to the closest reference object. The subsets can be splitted recursively if necessary.

However, all the algorithms mentioned above are static and the performance of the indexing algorithms deteriorate when more and more objects are inserted. In order to prevent performance degradation, expensive reorganization on the tree structure has to be done.

Dynamic Metric Space Indexing

There is not many metric space indexing methods proposed. The most well-known algorithm is M-Tree [16]. The structure of an M-Tree is similar to that of a R-Tree [26]. In the leave node of an M-Tree, the key of the corresponding objects, the object identifier and the distance from its parent are kept. A routing object is introduced in a non-leaf node for the indexing of its children objects. In a non-leaf entry of an M-Tree, the feature value of the routing object, the covering radius, together with the pointer to its parent object and the distance between the routing object and its parent is kept. All the children objects are guaranteed to be bounded within the covering radius from the routing object.

The M-Tree is built and maintained on a top-down fashion. Splitting of nodes will be done when node(s) overflow. Moreover, the two most common operations, range search and knn-search, are also applied on M-Tree. Thus, we proposed to use M-Tree as the indexing structure for indexing data generated on the previous chapters.

7.2 Clustering Algorithms on Metric Space

There are already many clustering algorithms proposed for clustering a set of data. However, there is no clustering algorithm that is designed for Clustering Metric Space data. In this section, we will introduce some clustering algorithms that is applicable on metric space data.

K-means [24] selects a set of cluster centers and the set of data points are assigned to the closest cluster center to form the cluster. The criterion for selecting which cluster the object belongs to is to minimize the overall distance. The cluster center can be shifted to the center of the set of clustered objects (the object with the minimal overall distance with all the objects in the cluster). However, the problem associated with k-means is that it has to pre-define the number of k (clusters to be formed) in order to get a good result on the clustering, which is not practical for most cases.

There are also some linkage-based clustering methods [9] proposed. The concept is simple: two objects are assigned to the same cluster if the distance between the two objects is less than a specified distance d. The two objects are said to be *connected* if their distance is less than d. Some modifications on this approach are proposed, which assign a minimal number of connected objects for the assignment of an object to a cluster [53]. Some clustering algorithms are also proposed by applying similar approaches, like DBSCAN [21].

7.3 The Integration of Clustering and Metric-Space Indexing Algorithm

In Chapter 6, we have already discussed the integration of R-Tree and the Incremental-DBSCAN clustering algorithm. We select Incremental-DBSCAN because this is an incremental clustering algorithm. Moreover, Incremental-DBSCAN applies R-Tree for the indexing and retrieval of relevant points. Hence, we need not use another indexing structure for indexing the clustered data points. Thus, we will also follow the same idea for the selection of the indexing and clustering algorithms.

Incremental-DBSCAN applies well on clustering vector-space data. However, since this clustering algorithm depends on only two variables *eps* and $MinPts$ [22], it is obvious that this algorithm should also work well on clustering metric space data. The problem is whether there exists an indexing algorithm that helps performing range search efficiently. M-Tree [16] is a suitable choice for the indexing metric space data. Additionally, the M-Tree indexing algorithm is quite similar to that of R-Tree [26]. Similar algorithms and analysis can be applied to the integration of M-Tree and Incremental-DBSCAN algorithms.

7.4 Proposed Algorithm

As mentioned in the previous section, similar logic based on IR-Tree is applied on the new indexing structure. The new structure, called IM-Tree, is based on the integration of M-Tree indexing algorithm [16] as well as Incremental DBSCAN clustering algorithm [22]. In this section, we will have a brief discussion on the changes on M-Tree for the integration.

7.4.1 Index Structure

As similar to the changes made in IR-Tree, the only modification made on the indexing structure is to add the counter for counting the number of data objects contained in its subordinates for the non-leaf nodes in an M-Tree. There is no need to make any modification to leaf nodes. For the structure of nodes in an M-Tree, please refer to [16].

Similar to the changes made on the clustering structure, a data structure is also added for keeping the clustering information. Please refer to section 6.3.1 for more detail on the data structure. Only one change should be made based on the generation of the cluster information. For IR-Tree, a centroid should be generated based on the set of cluster points in the center, which is the average of all the data points in it. However, in IM-Tree, it is difficult to generate the centroid based on the technique used in IR-Tree. For IM-Tree, the centroid should be selected from one of the cluster members. The one with minimal average distance from all the points within the cluster should be selected as the centroid and the *interval_radius* is computed based on the centroid.

7.4.2 Nearest Neighbor Search

The modification on the addition and deletion algorithm for IM-Tree is similar to the modifications made on IR-Tree. Please refer to section 6.3.2 and 6.3.3. for more details on the modification of the algorithms.

In fact, the algorithm is exactly the same as that used in IR-Tree. The generation of virtual radius based on the base radius and add-on radius is exactly the same as that in IR-Tree. The virtual radius is used as the input for the range search of IM-Tree. Since the count on the number of data records is kept in non-leaf node, the breadth-first knn-search can also be applied in IM-Tree.

7.5 Future Works

Although the basic concept of IM-Tree is more or less the same as that of IR-Tree, more research efforts should be placed on the issue. The main difference between vector space and metric space is that for the first case, any objects can be represented by an explicit vector (coordinate), but in metric space, it is hard to tell the explicit "location" of an object and only the relative distance between two objects can be done. This makes the computations of cluster centers and overlapping clusters difficult.

Moreover, in this chapter, only the feasibility of the IM-Tree is discussed. Hence, more experiments should be done in order to show the effectiveness of the algorithm.

Chapter 8

Conclusion and Future Works

Most of the traditional data manipulation algorithms handle structured data, like data on a relational database. However, as the size of the data set and the dimensionality of the data increases, new algorithms as well as modifications on old algorithms are proposed. The introduction of semi-structured data and OEM also arouse more attention on the data manipulation algorithms on semi-structured and structured data.

In this thesis, we presented algorithms on the manipulation of semi-structured data, by using web data as an example. We also presented an enhanced algorithm for indexing structured data on vector space. Based on our proposed algorithm, web documents can be classified and integrated based on OEM. Searching can be enhanced based on the representatives retrieved as well as the dictionary data structure. Moreover, based on the general structure generated from the set of web documents, the variation of web documents can be reduced.

We also proposed a modification on a traditional vector space indexing algorithm, R-Tree, which is the integration of the R-Tree indexing algorithm and a clustering algorithm Incremental-DBSCAN. Based on the integration of a clustering algorithm, an expensive knn-search can be transformed to a range search. Under a clustered environment based on the Incremental-DBSCAN, the performance of knn-search can be enhanced.

In this chapter, we will summarize our works and discuss on some further issues related to our works. In the coming section, we will conclude our works on semi-structured data manipulation. After that, we will summarize our works on the integration of vector-space indexing and clustering.

8.1 Semi-structured Data Manipulation

Semi-structured data are data having no rigid schema. The flexibility of semi-structured data allows users to represent their data in a more realistic form. However, the manipulation of semi-structured data is more difficult due to the lack of a rigid schema. Hence much research efforts are placed on the manipulation of semi-structured data.

Web data, which is characterized by having no rigid schema, is an example of semi-structured data. As the increasing popularity of the WWW, more and more data are placed on it. However, due to the variety on the format of web documents, it is difficult to retrieve the desired data from the WWW. In this thesis, we proposed an algorithm based on hyperlinks and the document semantics for the classification of web documents.

The enormous size of web documents prohibits us to analyze it as a whole. Therefore, in this thesis, we proposed a coarse partitioning algorithm by utilizing the hyperlinks embedded in web documents. Based on the number of in-links and out-links, we can classify web documents into different classes. Then based on the set of center nodes identified, documents can be partitioned into smaller subsets. After the coarse partitioning, we also proposed an algorithm for verifying the similarities within a subset of documents based on the similarity definitions proposed in this thesis. Unrelated documents are pruned away from the subset.

A representative document is retrieved for each subset. Moreover, the entries in each web document will be added into the dictionary data structure. The dictionary, together with the representative, can be used to enhance the searching of web documents.

However, the structure of web documents within the subset is still varied. We proposed the idea of general structure in this thesis. The generation of general structure is based on the extraction of frequently appeared OEM data structures within the subset. Then the subset of web documents are mapped under the general structure. The variation of data structure for the set of web documents within the subset can be reduced.

Although we showed in the experimental sections that our algorithm can help improving the searching and manipulation of web documents, further extensions on the proposed algorithm is still possible. In the proposed algorithm, it requires many many variables which should be pre-defined. The algorithm should be improved such that the set of parameters can be automatically adjusted based on the nature of the data set. Additionally, there are some issues not yet addressed in this thesis, like indexing and clustering of web data. With slight modification, our proposed algorithm can cluster and index the set of web documents.

8.2 Structured Data Manipulation

In the later chapters of this thesis, we proposed a new algorithm based on the traditional R*-Tree structure and Incremental-DBSCAN clustering technique. By using the clusters identified, an expensive knn-search can be transformed into a less expensive range search. Moreover, we also propose a breadth-first searching algorithm that can reduce the number of page access while performing knn-search. From the experimental results, we can show that our proposed algorithm out-performs the traditional algorithms on knn-search.

Our proposed algorithm can also reduce the effect of Dimensionality Curse. The problem of Dimensionality Curse is due to the poor construction on the indexing tree and highly overlapping of bounding boxes. However, in our proposed algorithm, we will prune away bounding boxes that do not contain the set of points we desire, by limiting the radius of searching or the nodes to examine.

We proposed our algorithm based on the R*-Tree indexing structure. However, our algorithm is not limited to the integration of Incremental-DBSCAN and R*-Tree only. Variants of R-Tree, like X-Tree and TV-Tree, can also be integrated with various clustering algorithms. Moreover, by using clustering technique, the information among closely-related data records can be maintained. Those information can be used for the spatial management of SAMs to improve the performance of those SAMs. More research effort will be put on this issue.

Besides, it is also possible to extend the idea to metric space. In this thesis, we also analyzed the feasibility for integrating a metric space indexing algorithm and clustering algorithm.

Bibliography

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. In *International Journal on Digital Libraries (1)*, pages 66–68, 1997.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 94–105, 1998.

[3] Naveen Ashish and Craig Knoblock. Wrapper Generation for Semi-structured Internet Source. In *Workshop on Management of Data*, 1997.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an Efficient and Robust Access Methods for Points and Rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

[5] S. Berchtold, C. Böhm, and H.-P. Kriegal. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 142–153, 1998.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegal. The X-tree: An Index Structure for High Dimensional Data. In *Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB)*, pages 28–39, 1996.

[7] Andre Bergholz and Johann Christoph Freytag. Matching Sehemata by utilizing Constraint Satisfaction Techniques. In *Workshop on Query Processing for Semistructured Data and Non-standard Data Formats*, 1999.

[8] T. Berners-Lee and D. Connolly. *HyperText Markup Language Specification – 2.0.* HTML Working Group of the IETF, 1995.

[9] H. H. Bock. *Automatic Classification.* Vandenboeck and Ruprecht, Gŏttingen, 1974.

[10] Daniel Boley, Maria Gini, Robert Gross, Eui-Hong (Sam) Han, Kyle Hastings, George Karypis, Vipin Kumar, Bamshad Mobasher, and Jerome Moore. Document categorization and query generation on the world wide web using webace. In *AAAI Review 13 (5-6)*, pages 365–391, 1999.

[11] T. Bozkaya and Z. M. Ozsoyoglu. Distance-based Indexing for High-dimensional Metric Spaces. In *ACM SIGMOD*, pages 357–368, 1997.

[12] S. Brin. Near Neighbor Search in Large Metric Spaces. In *Proc. 21th Int. Conf. on Very Large Data Bases (VLDB)*, pages 574–584, 1995.

[13] P. Buneman. Semistructured data. In *Proceedings on the Principles of Database System (PODS)*, pages 117–121, 1997.

[14] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180, 1996.

[15] T. Chiuch. Content-based Image Indexing. In *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*, 1994.

[16] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB Conference*, pages 426–435, 1997.

[17] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. In *Machine Learning 10*, pages 57–78, 1993.

[18] Alin Deutsch, Mary Fernandez, and Dan Suciu. Storing Semistructured Data With STORED. In *ACM SIGMOD*, pages 431–442, 1999.

[19] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[20] Jack Edmonds. Matroids and the greedy algorithm. In *Mathematical Programming, 1:126-136*, 1971.

[21] M. Ester, H-P Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB)*, pages 323–333, 1998.

[23] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD*, pages 163–174, 1995.

[24] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* San Diego, CA, Academic Press, 1990.

[25] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *Proc. 9th ACM Conference on Hypertext and Hypermedia*, pages 225–234, 1998.

[26] A. Guttman. R-trees: a Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[27] Joachim Hammer, Hector Gracia-Molina, Kelly Ireland, Yannis Papakonstantinoiu, Jeffrey Ullman, and Jennifer Widom. Information Translation, Medication, and Mosaic-Based Browsing in the TSIMMIS System. In *ACM SIGMOD Demo*, page 483, 1995.

[28] Eui-Hong (Sam) Han, George Karypis, , and Vipin Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *Technical Report # 99-019*, 1999.

[29] Eui-Hong (Sam) Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Clustering based on associatin rule hypergraphs. In *Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[30] http://cse.cuhk.edu.hk.

[31] http://www.acm.org.

[32] http://www.amd.com.

[33] Pearl J. Probaiilistic reasoning in intelligent systems: Networks for plausible inference. In *Morgan-Kaufmann*, 1988.

[34] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, 1997.

[35] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.

[36] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proc. Int. Conf. on Machine Learning*, pages 170–178, 1997.

[37] Kaufman L. and Rousseeuw P. J. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[38] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-Tree: an Index Structure for High-Dimensional Data. In *VLDB Journal*, volume 3, No. 4, pages 517–542, 1994.

[39] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. In *Neural Computation*, 1995.

[40] Y. S. Maarek and I. Z. Ben Shaul. Automatically organizing bookmarks per content. In *Proc. of 5th International World Wide Web Conference 28 (7-11)*, pages 1321–1334, 1996.

[41] Jason McHugh, Jennifer Widom, Serge Abiteboul, Qingshan Luo, and Anand Rajaraman. Indexing Semistructured Data. In *Technical Report in Standford University*, 1998.

[42] Jerome Moore, Eui-Hong (Sam) Han, Daniel Boley, Maria Gini, Robert Gross, Kyle Hastings, George Karypis, Vipin Kumar, and Bamshad Mobasher. Web page categorization and feature selection using association rule and principal component clustering. In *The 17th Workshop on Information Technologies and Systems*, 1997.

[43] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mininig. In *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*, 1994.

[44] Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, and Jennifer Widom. Querying Semistructured Heterogenous Information. In *Inter-*

national Conference on the Deductive and Object-oriented Database, pages 319–344, 1995.

[45] E. Schikuta. Grid Clustering: An efficient Hierarchical Method for Very Large Data Sets. In *Proc. 13th Conf. on Pattern Recognition, Vol. 2*, 1996.

[46] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Database. In *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB)*, pages 428–439, 1998.

[47] J. R. Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain.* http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient-figs.ps, 1994.

[48] D. Titterington, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions.* John Wiley & Sons, 1985.

[49] J. K. Uhlmann. Satisfying General Proximity/Similarity Queries with Metric Trees. In *Inf. Proc. Lett. 40(4)*, pages 175–179, 1991.

[50] W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proc. 23th Int. Conf. on Very Large Data Bases (VLDB)*, pages 186–195, 1997.

[51] Ron Weiss, Bienvenido Vélez, Mark A. Sheldon, Channathip Namprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. Hypursuit: A hierarchical network search enging that exploits contenmt-link hypertext clustering. In *Seventh ACM Conference on Hypertext*, pages 180–193, 1996.

[52] D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *Proc. 12th Int. Conf. on Data Engineering, New Orleans, USA*, 1996.

[53] D. Wishart. Mode Analysis: A Generalization of Nearest Neighbor, which reducing Chaining Effects. In *in A. J. Cole(Hrsg.), 282-312*, 1969.

[54] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. In *Proc. 14th Int. Conf. on Data Engineering (ICDE'98)*, pages 324–331, 1998.

[55] Jihoon Yang, Prashant Pai, Vasant Honavar, and Les Miller. Mobile intelligent agents for document classification and retrieval: A machine learning approach. In *Proceedings of the Fourteenth European Meeting on Cybernetics and Systems Research (EMCSR'98)*, 1998.

[56] Clement Yu, King-Lup Liu, Weiyi Meng, Zonghuan Wu, and Naphtali Rishe. Finding the Most Similar Documents across Multiple Text Databases. In *SIGMOD Record 26(4)*, pages 8–15, 1997.

[57] T. Zhang, R. Ramakrishnan, and M. Livny. An Efficient Data Clustering Method for Very Large Databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 103–114, 1996.