A Multi-level Hierarchical Index Structure for Supporting Efficient Similarity Search on Tag Sets

Jia-Ling Koh and Nonhlanhla Shongwe
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan
jlkoh@csie.ntnu.edu.tw

Chung-Wen Cho
Cloud Computing Center for Mobile Applications Industrial Technology Research Institute
Hsinchu, Taiwan
chungwen.cho@gmail.com

***Abstract*—Social communication websites has been an emerging type of a Web service that helps users to share their resources. For providing efficient similarity search of tag set in a social tagging system, we propose a multi-level hierarchical index structure to group similar tag sets. Not only the algorithms of similarity searches of tag sets, but also the algorithms of deletion and updating of tag sets by using the constructed index structure are provided. Furthermore, we define a modified hamming distance function on tag sets, which consider the semantically relatedness when comparing the members for evaluating the similarity of two tag sets. This function is more applicable to evaluate the similarity search of two tag sets. A systematic performance study is performed to verify the effectiveness and the efficiency of the proposed strategies. The experiment results show that the proposed MHIB approach further improves the pruning effect of the previous work which constructs a two-level index structure. Especially, the MHIB approach is well scalable with respect to the three parameters when using either the hamming distance or the modified hamming distance for similarity measure. Although the insertion operation of the MHIB approach requires higher cost than the naïve method, with the assistant of the constructed inverted list of clusters, it performs faster than the previous work. Besides, the cost of performing deletion operation by using the MHIB approach is much less than the other two approaches and so is the update operation.**

Keywords: Social tagging;index structure;similarity search.

I. INTRODUCTION

Social tagging has became widely popular in various applications, such as photo-sharing sites (e.g Flickr), video-sharing sites (e.g. YouTube), and social bookmarking sites (e.g. CiteULike and Delicious). Social tagging systems enable users to freely annotate web resources of interest with tags for describing the content or semantics of the resources. Besides, a resource can be tagged by more than one tag. It is also possible that a tag is used to label different resources. These tags can be used to search the web resources by users conveniently.

In general, if two web resources are assigned the similar sets of tags, we believe that they should be semantics-related. Instead of comparing the content of web resources, the similarity measure of tag sets provides a good way of finding similar resources, which is an essential operation required to support resource recommendation. However, the amount of various tag sets in a social tagging system is usually huge. Accordingly, providing efficient similarity search on tag sets is a critical problem in a social tagging system.

Given a set of tags as a query, a distance measure function, and a query threshold, the goal of a similarity search is to find all web resources whose tag sets have distances with the query less than the given query threshold. To efficiently find the semantics related web resources with a given tag set, there is a need for constructing an index structure of web resources according to their tag sets. The goal of the index structure is well organizing the web resources to provide efficient similarity searches of tag sets.

A transaction in a supermarket database indicates the set of items purchased by a customer. For a given transaction of a customer, the similar transactions of other users found from the transaction database can be selected as recommend items that the customer may be interested. Similarity search in a transaction database thus becomes one of the important techniques for data analysis in data mining [1]. Let the set of tags given to a web resource correspond to a transaction, the indexing method for supporting similarity search on a transaction database is thus applicable for providing similarity search of tag sets on a social tagging system.

A method called the two level bounding mechanisms was proposed in [6], which is a novel indexing method to support similarity search in a dynamically updated transaction database. The incoming transactions are incrementally grouped into clusters. Each cluster is characterized by two features: the outer border and the inner border, where the outer border and inner border is the union and the intersection of all the transactions in the cluster respectively. The cardinality of the set difference between the two borders is used to evaluate the similarity among the transactions in a cluster. In each cluster, the transactions are further grouped into sub-clusters called batches. All the transactions are thus organized as a two-level index structure where the distance between each pair of transactions in a cluster is bounded by a given cluster threshold. To the best of our knowledge, the two level bounding mechanisms are better than most of the other approaches in dealing with the similarity search problem on transaction databases. However, this approach has some drawbacks for improvement. First, there is no limitation on the inner border of a cluster. Consequently, it is possible that a set of small and disjoint transactions are grouped into a cluster when the cardinality of their union is

less than the cluster threshold, which results in less effectiveness of the pruning strategy on performing similarity searches. Secondly, the proposed index structure provides only one level of transaction cluster. Besides, a batch's similarity quality within a cluster is not specially controlled. Consequently, the above drawbacks make the index structure to be less effective for efficiently searching similar sets in some cases.

Furthermore, because users have a flexible way of giving tags in a tagging system, users can describe the similar concept in different ways. For example, for a picture which shows a cat, most users would use the tag 'cat', where others would use a tag like 'kitty' or 'pet'. However, these tags are semantically related. Therefore, it will greatly limit the performance of similarity search of tag sets in tag space if the comparison of members in two tag sets uses exact matching of tags.

Our work is motivated from the challenges mentioned above. Therefore, the main goal of this paper is to further improve the searching efficiency of the indexing strategy proposed in [6]. Besides, we would like to apply the index structure to support similarity search of tag sets in a tagging system. The contributions of this paper are summarized as follows:

(1) We propose a multi-level hierarchical index structure to group similar tag sets. The proposed strategies not only support static similarity searches of tag sets, but also support the insertion, deletion, and updating operations of tag sets.

(2) A modified hamming distance function is defined on tag sets. Therefore, when comparing the members for evaluating the similarity of two tag sets, two semantically related tags have some degree of matching. This function is more applicable to evaluate the similarity search of two tag sets.

(3) A systematic performance study is performed to verify the effectiveness and the efficiency of the proposed strategies.

The rest of the paper is organized as follows. The related works are surveyed in Section 2. The proposed multi-level hierarchical index structure with bounding mechanism is introduced in Section 3. Then we define the modified hamming distance and provide the corresponding modification of similarity searching strategy in section 4. The performance evaluation on the proposed index structure and processing algorithms is reported in Section 5. Finally, we conclude this paper and show the directions for our future studies in Section 5.

II. RELATED WORKS

A. *Research Issues on Social Tagging System*

Recently, collaborative tagging system has grown in popularity on the web, which allows users using tags to tag their favorite online documents, photographs, and other content. Various research issues on the social tagging systems have been studied. The semantic tag clustering search was proposed in [14] to solve the problem of syntactic and semantic tag variations during user search and browse activities. A framework for social bookmark weighting was proposed in [7] to estimate the effectiveness of each of the bookmarks for improving social search effectiveness. The strategy is also applicable for tag recommendation, user recommendation, and document recommendation. The tag prediction problem was studied in [5]. A tag inference technique was proposed to assign tags to un-tagged documents according to the frequency with which a tag appears in a given document neighborhood, the frequency with which a pair of tags co-occurs in a document, the distance between a tag and a given document in the resource graph, and the similarity between the documents the tags are attached to.

Among the different studies on social tagging system, discovering the implicit relations between the web resources can provide recommendation of semantics related resources when users accessing a resource. Accordingly, [18] calculated the pair-wise similarity between tagged documents to discovering user-induced links between documents. As mentioned in [18], one of the approaches to discover the implicit links between resources in a social tagging system is to calculate the similarity between the sets of tags assigned to the resources. Accordingly, similarity searching of web resources is an elementary and important operation to support further application on social tagging systems.

B. *Similarity Search*

Similarity search is an important function required for various applications, such as clustering of transaction data [12] and duplicates detection of web documents [10]. A quantitative way to define two objects are similar is to give a distance function or a similarity function. Here we consider the objects whose features are described by a set of tokens. Accordingly, the similarity of two objects is evaluated as the similarity between their corresponding sets of tokens.

Hamming distance is a widely used distance function for sets. Given two sets of tokens, S1 and S2, the hamming distance between S1 and S2 is defined as the cardinality of their symmetric difference. That is $Hdist(S1, S2) = |(S1-S2)\cup(S2-S1)|$, which is equivalent to $|(S1-S2)|+|(S2-S1)|$. A small hamming distance score between a pair of records implies they have high similarity. In [6], a novel indexing method for similarity search of transaction data was provided, where hamming distance was used to evaluate the similarity between sets of items. Moreover, hamming distance was used in [13] to evaluate the similarity of signatures for constructing hierarchical clusters of signatures. The problem for efficiently performing similarity join with edit distance constraints was studied in [15]. By analyzing the locations and contents of mismatching q-grams, two new edit distance lower bounds were derived to reduce the candidate sizes and save computation time.

The similarity function measures the degree of similarity between two objects, which returns a similarity value in [0, 1]. A higher similarity value of objects indicates that they are more similar to each other.

There are a number of similarity functions that can be used to evaluate the similarity between two objects consisting of sets of feature tokens, such as Jaccard similarity, cosine similarity, and overlap similarity [17]. Jaccard similarity is commonly used to evaluate the similarity of categorical data. Given two sets of tokens S1 and S2, the jaccard similarity between S1 and S2 is defined as $Jacc\_sim(S1, S2) = (|S1 \cap S2| / |S1 \cup S2|)$. Jaccard similarity was used in [17] to perform duplicate detection of web documents. In [2], the author studied the asymmetric version of Jaccard similarity to provide indexing strategy for Jaccard containment similarity function. In [3] , both the hamming distance and Jaccard similarity function were used to identify all pairs of sets that are highly similar for performing a set-similarity join. On the other hand, from a sparse set of vectors, the cosine similarity is used in [4] to find all pairs of vectors whose similarity score is above a given threshold. Also consider the various similarity functions of sets, instead of giving a query threshold, [16] considered the problem of finding the top-k pairs of records ranked by their similarities.

C. Index Structures for Similarity Search of Objects

For providing efficient similarity search of objects consisting of sets of tokens, it is necessary to design an index structure used for effectively pruning the search space when processing a query. The concept of signature has been widely used in the studies of index strategies for market basket data or sets and categorical data, such as [1], [4], [9], [11], [13]. The index strategies using signatures are mainly classified into signature table approaches and signature tree approaches.

The signature table approach [1] first partitions all the items in the transaction database into K groups of frequently correlated items. Then a bit vector with K bits is used to represent a transaction, where each bit is associated with one of the K groups of items called vertical signatures. If a transaction contains a sufficient number of the items in a group, the corresponding bit of its signature is set to 1. Otherwise, the bit is set to 0. Accordingly, the transactions are hashed into a signature table consisting of 2^k entries according to their signatures. By comparing the signatures of the query and each signature table entry, the estimated lower bound of the distance between the query and the transactions hashed into the entry is used to prune the search space for finding similar transactions. Therefore, this approach performs efficiently for nearest neighbor search queries. Instead of grouping correlated items globally to form vertical signatures, [8] considered the problem that different subsets of the data may have different data distribution and proposed the localized signature table for blocks consisting of highly related transactions. However, the performance of the SG-table approach is sensitive to various parameters which have to be tuned in advance. Besides, the highly correlative sets of items, which determine the vertical signatures of the signature table, may change from time to time. Accordingly, the SG-table approach is not suitable for a dynamic database.

The signature tree was proposed in [9], which is a dynamic balanced tree similar to an R-tree for maintaining the signature vectors. For a leaf node entry in the signature tree, it contains the signatures of transactions within a disk page and their transaction ids. Each internal node contains a number of entries and each entry contains a signature and a pointer linking to a child node, where the signature is the logical OR of all the signatures in the child node. A signature representation method and algorithms were proposed in [11] to guarantee getting the complete results for similarity searching in transaction databases. Although the R-tree based approach provide the flexibility of index updates, this approach has several disadvantages. First, for the internal nodes at a higher level, more transactions contained in their sub-trees results in most bits in their signatures are 1. Consequently, it is not distinguishable whether the transactions stored in the descendants of the node are similar to the query. Besides, the time complexity of processing node splitting in the tree structure is high.

For providing efficient similarity search in a transaction database which is frequently updated, [6] proposed an index structure where the similar transactions are grouped into clusters and the transactions in a cluster are divided into disjoint groups called batches. When processing a similarity search for a given query, the lower bound and upper bound on distance between the query and the transactions assigned within a cluster can be estimated according to the recorded information. Consequently, if the estimated distance lower bound is larger than the query threshold or the upper bound is less than the query threshold, it implies that all transactions in the cluster can be pruned or all transactions are similar to the query without actually computing their distances to the query. The strategy is also applicable for the batch level. The experiment results show that this approach performs more efficiently than the SG-Tree approach [9] on the execution times of similarity search and index updating.

D. Two-level Bounding Mechanism

One goal of our work is to further improve the indexing strategy previously proposed by [6], which is provided for supporting similarity search in transaction databases. Therefore, we briefly introduce the method in this section.

Let $I = \{i_1, i_2, \ldots, i_m\}$ denote the set of items in a specific application domain. Let *TDB* denote a database of transactions, where each transaction T_i in *TDB* is a non-empty subset of *I*. In [6] the transactions in *TDB* are incrementally grouped into clusters. Each cluster *C* in the index structure is characterized by two features, the outer border of *C* and the inner border of *C*, denoted as *C.Co* and *C.Ci*, respectively. The *C.Co* stores the union of all the transactions in cluster *C* and *C.Ci* stores the intersection of all the transactions in cluster *C*. Thus, *C.Ci* is a subset and *C.Co* is a superset of each transaction in the cluster *C*, respectively. The *difference* of a transactions T_i with respect to another transaction T_j is denoted as $diff(T_i, T_j)$, which is defined to be $|T_i - T_j|$. Furthermore, the hamming distance is

used to evaluate the distance between T_i and T_j, which is defined as $dist(T_i, T_j) = diff(T_i, T_j) + diff(T_j, T_i)$. When constructing the index structure, a cluster threshold ε is used to limit that the distance between the outer border and the inner border of a cluster is not greater than ε. Accordingly, the distance of each pair of transactions within a cluster is controlled within an upper bound ε.

The transactions in a cluster are further grouped into batches, where all the transactions in a batch *B* have the same number of items. Accordingly, *diff*(*C.Co*, *T*) is unique for each transaction T in a batch *B* of a cluster *C*, and so is *diff*(*T*, *C.Ci*). Each batch *B* in a cluster *C* is thus characterized by two features, *dvo*(*C*, *B*) and *dvi*(*C*, *B*), which stores the values of *diff*(*C.Co*, *T*) and *diff*(*T*, *C.Ci*), respectively. These two features are called the difference-value pair of *B*. For example, Figure 1 shows the content of a cluster *C* in the index structure, which contains the transactions {*a*, *b*, *c*}, {*a*, *b*}, {*b*, *c*} and {*b*}. The outer border *Co* and inner border *Ci* of *C* are {*a*, *b*, *c*} and {*b*}, respectively. The four transactions are further grouped into batches B_1, B_2 and B_3 in *C* based on the number of items in the transactions. The difference-value pair associated with each batch is represented in the header of the batch.

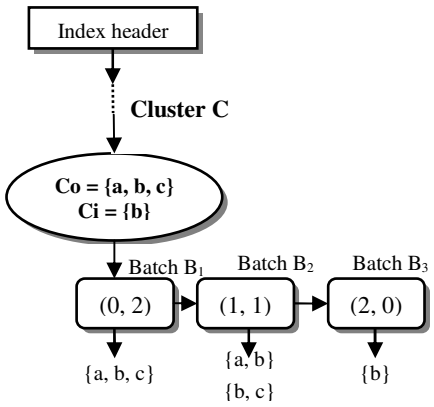


Figure 1. A sample cluster and its batches.

A two level bounding mechanism is proposed in [6] to search the transactions whose distance with the query transaction T_q is less than a given query threshold δ from the index structure. The two borders of a cluster are used to estimate the upper bounds and lower bounds of the distance between the query transaction and the transactions in the cluster. The first level bounding mechanism is used to decide to prune or return all the transactions in a cluster. Similarly, the difference-value pair of a batch is used to estimate the upper bounds and lower bounds of the distance between the query transaction and the transactions in the batch. If it is not decidable according to the first level bounding mechanism, the second level bounding mechanism is triggered to prune or return transactions at the batch level.

The details of the two level bounding mechanisms are described as following. Given a query transaction T_q, each cluster *C* in the index structure is checked one by one. The first level bounding mechanism estimates a lower bound of the distance between T_q and any transaction in *C*, which is called the first level lower bound and denoted as 1st-LB (T_q, *C*). An upper bound of the distance between T_q and any transaction in *C* is also estimated, which is called the first level distance upper bound and denoted as 1st-UB (T_q, *C*). The equations used to estimate 1st-LB(T_q, *C*) and 1st-UB(T_q, *C*) are defined below.

$$1\text{st-LB}(T_q, C) = diff(T_q, C.Co) + diff(C.Ci, T_q) \quad \ldots\ldots\ldots(1)$$

$$1\text{st-UB}(T_q, C) = diff(T_q, C.Ci) + diff(C.Co, T_q) \quad \ldots\ldots\ldots(2)$$

By comparing 1st-LB (T_q, *C*) or 1st-UB (T_q, *C*) with the query threshold δ, there are three possible cases. The corresponding processing for the three cases is described as follows.

1) 1st-LB(T_q, *C*) > δ: all the transactions in cluster *C* can be pruned from the search space.

2) 1st-UB(T_q, *C*) <= δ: all the transactions in cluster *C* are retrieved as the results.

3) Neither of the above: the second level bounding mechanism is triggered to check each batch *B* in cluster *C*.

If 1st-LB(T_q, *C*) is greater than δ, it implies that the distances of all the transactions in cluster *C* with T_q must be greater than δ. Therefore, none of the transactions in the cluster will be an answer. That is why all the transactions can be pruned from the search space in this case. If 1st-UB(T_q, *C*) is less or equal to δ, all the transactions in the cluster are similar to the query within distance δ. Therefore, all the transactions are returned as part of the answers. If neither of the above two cases holds, some transactions may satisfy the similarity search but the others may not. Thus, for each batch *B* in cluster *C*, the second level bounding mechanism is performed.

The second level distance lower bound, denoted as 2nd-LB(T_q, *C*, *B*), and the second level distance upper bound, denoted as 2nd-UB(T_q, *C*, *B*), are estimated by the following equations 3 and 4, respectively.

$$2\text{nd-LB}(T_q, C, B) = diff(T_q, C.Co) + |diff(C.Co, T_q) - dvo(C, B)| \quad \ldots(3)$$

$$2\text{nd-UB}(T_q, C, B) = dist(T_q, C.Ci) + dvi(C, B) \quad \ldots\ldots\ldots\ldots(4)$$

Similar to the first level bounding mechanism, the second level bounding mechanism has three possible cases of processing as follows.

1) 2nd-LB(T_q, *C*, *B*) > δ: all the transactions in batch *B* can be pruned from the search space.

2) 2nd-UB(T_q, *C*, *B*) <= δ: all the transactions sets in batch *B* are retrieved as the results.

3) Neither of the above: Check *dist*(T_q, T) for each transaction T in batch *B*. If the distance is less than δ, T is returned as an answer.

According to the processing of the first-level bounding mechanism, the similarity search can be performed efficiently when case 1 or case 2 occurs because it reduce the times of performing distance checking for all the transactions in a cluster. Likewise, the second-level bounding mechanism achieves the similar effect.

III. A MULTI-LEVEL HIERARCHICAL INDEX STRUCTURE WITH BOUNDING MECHANISM

We propose a multi-level hierarchical index structure which supports the similarity search of sets. Here, we assume that the main application scenario is for indexing the tag sets in a social tagging system. Similar to the index structure proposed in [6], the tag sets are grouped into clusters. However, a cluster can have sub-clusters. The tag sets in a leaf-cluster are further grouped into batches.

A constructing algorithm of the multi-level hierarchical index structure is provided for inserting the tag sets of the objects in a database into the index structure. Once a query is given, we apply the two level bounding mechanisms to search the index structure and return the objects having similar tag sets with the query efficiently. In addition, in order to support dynamic updates of the database, we also propose the necessary algorithms for maintaining the index structure, which includes deletion of existing tag sets, insertion of new tag sets, and updating of existing tag sets. The details of the proposed index structure and processing algorithms are introduced in the following subsections.

A. Terms Definition

Let *WDB* denote a database of web resources, where each object O_i in *WDB* has a set of annotated tags, denoted as O_i.tagset. The proposed multi-level hierarchical index structure organizes the sets of tags for all the objects in *WDB* into a forest-like structure, where each tree corresponds to a cluster. Let *C* denote a cluster in the index structure, which is characterized by two features: the outer border of *C* and the inner border of *C*, denoted as *C.Co* and *C.Ci*, respectively. The *C.Co* stores the union of all the transactions in cluster *C* and *C.Ci* stores the intersection of all the transactions in cluster *C*.

The similarities of the tag sets stored within a cluster *C* is controlled by limiting the maximum distance between its outer border(*C.Co*) and inner border(*C.Ci*). Here, the distance between two tag sets T_i and T_j are evaluated by computing $dist(T_i, T_j)$.

We define two different thresholds to control the similarity of the tag sets within the clusters at different levels.

1) *maxd_root*: It is used to limit the maximum distance between the outer and inner borders of a cluster *C* at the root level in the index structure.

2) *maxd_leaf*: It is used to limit the maximum distance between the outer and inner borders of a cluster *C* at the leaf level in the index structure.

In the cluster at the leaf level, the tag sets are further grouped into batches. Let *B* denote a batch in the index structure, which is also characterized by two features: the outer border of *B* and the inner border of *B*, denoted as *B.Bo* and *B.Bi*, respectively. The *B.Bo* stores the union of all the transactions in cluster *B* and *B.Bi* stores the intersection of all the transactions in cluster *B*. Therefore, we defined the third threshold value *maxd_batch* to the maximum distance between the outer and inner borders of a batch *B*.

B. Index Structure Construction

In this section, we will explain how the index structure is constructed for the given tag sets of the dataset *WDB*.

1) Index Construction .

Initially, the index structure is empty. The process of index construction is to insert the tag set O_i.tagset into the index structure for each object O_i in *WDB* one by one.

Let T_i denote O_i.tagset. When inserting a tag set T_i into the index structure, first, we have to check if the tag set already exists in the index structure. The checking is performed by the searching strategy introduced later. If the tag set T_i is found from the index structure, it means that there exists another object in *WDB* annotated with the same set of tags. Therefore, the address of the object O_i is added into the object list of the tag set in the index structure. If the tag set T_i is not in the index structure, it is necessary to find a cluster or create cluster for inserting T_i into.

For a cluster *C* at the root level (first level) of the index structure, T_i is allowed to be inserted into *C* if $C.Ci \cap T_i \neq \phi$ and $dist(new\_Co, new\_Ci) \leq maxd\_root$, where $new\_Co = (C.Co \cup T_i)$ and $new\_Ci = (C.Ci \cap T_i)$. In other words, the distance between the new outer border and inner border of *C* after inserting T_i still satisfies the *maxd_root* constraint. Besides, because the inner border of each cluster is required to be non-empty, it prevents a set of small and disjoint transactions being grouped into a cluster.

If there were a number of clusters at the root level the tag set T_i can be inserted into, we will choose the cluster C_{best} which has the smallest difference between the new two borders after inserting T_i. T_i is inserted into cluster C_{best} and the borders of C_{best} are updated as $C_{best}.Co = (C_{best}.Co \cup T_i)$ and $C_{best}.Ci = (C_{best}.Ci \cap T_i)$. If the cluster C_{best} has sub-clusters, the similar processing is performed recursively to find a sub-cluster of C_{best} for inserting T_i until T_i is inserted into a cluster at the leaf level.

Let C_{leaf} denote the cluster at the leaf level that T_i is inserted into. According to the cardinality of T_i, T_i is inserted into the corresponding batch under C_{leaf}. If the batch does not exist, a batch B is created to store T_i, whose difference-value pair is set to be ($dist(C_{leaf}.\text{Co}, T_i)$, $dist(T_i, C_{leaf}.\text{Co})$). Besides, both the borders of C_{leaf} and *B* are updated as follows:

$C_{leaf}.Co = (C_{leaf}.Co \cup T_i)$ and $C_{leaf}.Ci = (C_{leaf}.Ci \cap T_i)$;

$B.Bo = (B.Bo \cup T_i)$ and $B.Bi = (B.Bi \cap T_i)$.

The two borders and the difference-value pairs of the other batches under this cluster are also updated accordingly.

After inserting T_i into C_{leaf}, the cluster C_{leaf} is checked if $diff(C_{leaf}.Co, C_{leaf}.Ci)$ is greater than *maxd_leaf*. If the insertion of T_i into C_{leaf} making C_{leaf} violates the *maxd_leaf* constraint, the splitting algorithm described in the next subsection is performed.

If there is no cluster at the root level which T_i can be inserted into, a new cluster *C'* is created. T_i is inserted into cluster *C'* and the borders of *C'* are $C'.Co = T_i$ and $C'.Ci = T_i$. Besides, a batch *B'* under the cluster *C* is created for storing

T_i, whose difference-value pair is set to be (0, 0). Besides, $B'.Bo = T_i$ and $B'.Bi = T_i$.

2) Batch Splitting

A cluster C_{leaf} at the leaf level is splitting into sub-clusters when the difference between its outer and inner borders violates the *maxd_leaf* constraint. Then one additional level of clusters is generated, which includes the newly created leaf clusters under C_{leaf}. The batch splitting consists of two phases: the batch separating phase and the batch merging phase.

I) Batch separating phase

In the batch separating phase, for each batch B under cluster C_{leaf}, $dist(B.Bo, B.Bi)$ is calculated to check whether the distance is greater than threshold *maxd_batch*. If the batch B violates the *maxd_batch* constraint, the batch is separated into two new batches as follows. We first calculate the distance of each pair of tag sets in the batch; the pair of tag sets with the largest distance is selected to be seeds of the two new batches B_1 and B_2 respectively. Each of the rest of the tag sets in B is then assigned to one of the two batches according to the distances between the tag set and the borders of the batches. The batch which has the shortest distance to the tag set is chosen for assigning that tag set into. Once a tag set is assigned to one new batch, the new borders of the batch are calculated.

II) Batch merging phase

After the batch separating phase is complete, all the batches under the cluster satisfy the *maxd_batch* constraint. The distance between each pair of batches is calculated. The pair of batches with the largest distance is selected as the seeds which are assigned into two new clusters C_1 and C_2 respectively. Each of the rest batches is then assigned to one of the two clusters, which has the smallest difference with the batch according to its borders. After the assignment is finished, the difference value pairs of the batches need to be updated. This process results in two new sub-clusters C_1 and C_2 under C_{leaf}.

For example, suppose that the three thresholds for constructing the index structure are as follows: *maxd_root* = 5, *maxd_leaf* =3 and *maxd_batch*= 2. A cluster in the index structure contains the tag sets {a, b, c, d, e}, {b, c, d, e}, {a, d, e}, and {c, d, e}. After inserting the tag sets {a, b, c, d}, the batches in the cluster is shown as Figure 2(a). Since the cluster border violates *maxd_leaf*, the batch border of each batch under this cluster is checked if they violet α. In this case, the distances between the batch borders of all the three batches are less or equal to α. So, no batches need to split. The three batches are then merged to form the two new sub-clusters C_1 and C_2 as shown in Figure 2(b).

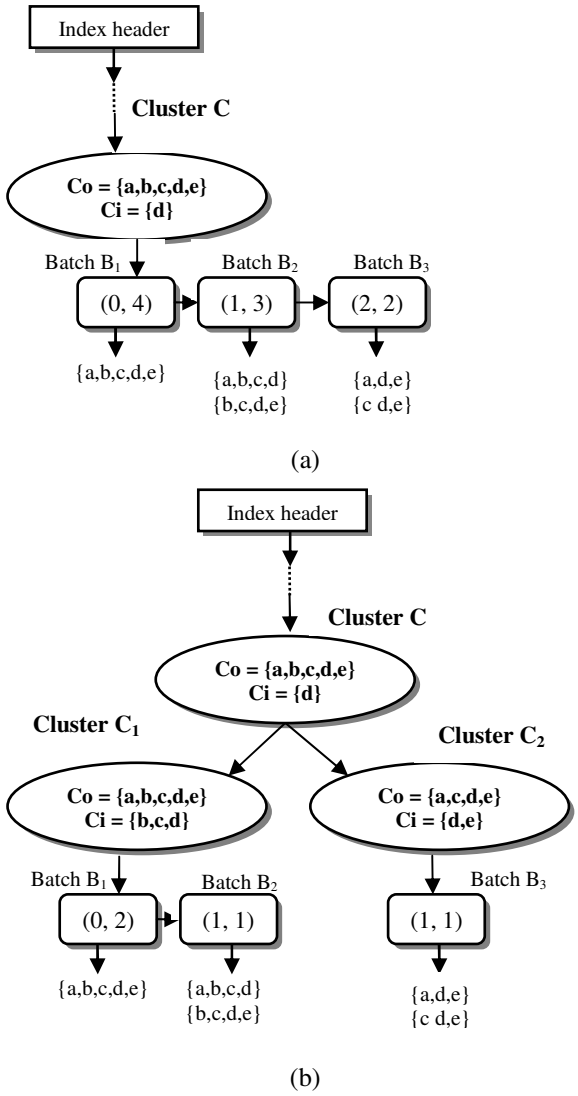


Figure 2: Example of batch splitting.

3) Inverted List Construction

The inner border of each cluster is required to be non-empty when constructing the index structure. Accordingly, an inverted list is constructed for maintaining the clusters at the root level. The inverted list has the corresponding entries for all the tags. For each corresponding tag, the inverted list keeps a list of cluster ids for the clusters whose inner border contains the tag. For example, the id of the cluster C shown in Figure 2(b) is maintained in the list for tag **d**.

The inverted list is mainly used to enhance the performance of the insertion and deletion operations. When inserting a tag set into the index structure, the inverted list is used to efficiently find candidate clusters with an inner border containing at least one tag in the inserted tag set. When performing a deletion operation on a tag set, it requires searching the index structure to find the deleted tag set. Because the searching requires exact match, the query threshold δ should be zero. The inverted list is thus used to efficiently return the clusters that might contain the tag set to be deleted. Accordingly, it prevents from checking all the

clusters at the root level. Therefore, the inverted list is also applicable for speeding up the searching of tag set when the query threshold δ is zero. The detailed processing methods of searching and deletion operations are described as the following subsections.

4) Similarity Search Operation

Given a query T_q which consists of an non-empty set of tags and a specified query threshold (δ), the searching algorithm is used to find the objects in *WDB* where the distances between their tag sets and T_q are less than or equal to δ. When the given search threshold (δ) is small, it means that the user is interested in the objects with highly similar tag sets to the query. If the search threshold (δ) is set to 0, only the objects have the same set of tags will be returned.

For each cluster C at the root level, we calculate 1st-LB(T_q, C) according to equation(1). If the value of 1st-LB (T_q, C) is greater than δ, it implies that all the tag sets within C have distance with T_q greater than δ. Thus, all the tag sets within C can be pruned without needing further checking. Otherwise, the 1st-UB(T_q, C)) is calculated according to equation(2). If the value of 1st-UB(T_q, C)) is less than or equal to δ, it is indicated that all the tag sets within C have distance with T_q not greater than δ. Accordingly, all the objects with the tag sets in the cluster C are returned as answers.

If neither one of the checking on the two bounds satisfies, we have to check the clusters at the next level under C. If C is not a cluster at the leaf level, the same bounding mechanism checking is performed recursively on each sub-cluster of C. The same processing repeats until a cluster C_{leaf} at the leaf level is reached, which consists of batches. On the batch level, the checking on the 2nd-LB(T_q, C_{leaf}, B)) is used to check whether all the tag sets in a batch B can be pruned from the search space; else the checking on the 2nd-UB(T_q, C_{leaf},, B) is performed to decide whether all the tag sets in the batch are part of the answers. If the checking on the 2nd-UB(T_q, C_{leaf}, B) also fails, the distance between T_q and each tag set T in the batch B is calculated. If the value of dist(T_q, T) is not greater than δ, the tag set T is returned as a query result.

In the case when the query threshold is set to 0, it means that the user requires the objects have the same set of tags as T_q. The clusters that are needed to be checked are only the clusters whose inner border contains any tag in T_q. Therefore, for each tag t in T_q, the list of cluster ids for tag t in the inverted list is retrieved to find a set of candidate clusters for searching. This strategy significantly enhances the pruning efficiency of searching space when performing the exact match search.

5) Deletion Operation

When an object O_d in *WDB* is deleted, the deletion operation on its tag set T_d needs to be performed. First, a search for T_d with query threshold 0 is performed. We have to check whether only one object in *WDB* has the tag set T_d. If there is more than one object in the object list of tag set T_d, we only need to remove the object O_d from the object list of T_d and do not need to delete T_d from the index structure. Otherwise, the tag set T_d is removed from the index structure and the borders of the batch and all the clusters that T_d belongs to should be updated accordingly.

6) Update Operation

In a tag database, sometimes users want to modify the tag set of an object such as add a tag or remove a tag from the tag set. It can be performed by deleting the existing tag set and inserting the new tag set after modification.

IV. MODIFIED HAMMING DISTANCE

A. Definition of Modified Hamming Distance

According to the hamming distance defined previously, two semantic-related concepts represented by two different tags will be considered different concepts. It will reduce the effectiveness of similarity search of tag sets in the social tagging system. Therefore, we define a new distance measure called the modified hamming distance in this section, which is an extension of the hamming distance measure. When computing the cardinality of the intersection between two tag sets, we consider not only the number of same tags in the two tag sets but also the degrees of semantic relatedness of the unmatched tags in both tag sets.

According to the definition of hamming distance between two tag sets T_i and T_j given in the subsection of terms definition, we can derive the following result:

$$\begin{aligned} dist(T_i, T_j) &= diff(T_i, T_j) + diff(T_j, T_i) \\ &= (|T_i|-|T_i \cap T_j|)+(|T_j|-|T_i \cap T_j|) \\ &= |T_i|+|T_j|-2|T_i \cap T_j| \end{aligned}$$

In other words, the hamming distance between two tag sets T_i and T_j is mainly determined by the cardinality of both T_i and T_j and the cardinality of their intersection.

We define the modified hamming distance function to be:

$$mdist(T_i, T_j) = |T_i|+|T_j|-2(|T_i \cap T_j|+SR(T,T_q) \quad \ldots\ldots\ldots\ldots(5),$$

where $SR((T,T_q))$ computes the degree of semantic relatedness among the un-matched tags in T and T_q .

We use the correlation measure between two tags t_a and t_b to represent their degree semantic relatedness as follows:

If $correlation(t_a, t_b)>0$

$$related\text{-}degree(t_a, t_b) = correlation(t_a, t_b);$$

Otherwise, $related\text{-}degree(t_a, t_b) = 0$.

Let k denote the value of min($|T-T_q|$, $|T_q-T|$). The $SR(T,T_q)$ is defined to be the maximum value of the sum of the related-degrees of k disjoint pairs between (T-T_q) and (T_q-T).

To prevent from enumerating all possible k disjoint pairs between (T-T_q) and (T_q-T), we use a greedy approach to approximately estimate the value of $SR(T,T_q)$ as follows. Let T_1 denote (T-T_q) and T_2 denote (T_q-T). First, for t_a in T_1 and t_b in T_2, the pair (t_a, t_b) with the maximum $related\text{-}degree(t_a, t_b)$ is selected. Then t_a and t_b are removed from T_1 and T_2,

respectively. Then the same processing is performed to select the next pair with the maximum *related-degree*(t_a, t_b) from T_1 and T_2 until k disjoint pairs are selected. Then $SR(T,T_q)$ is estimated by the sum of the related-degrees of these selected k disjoint pairs.

We use an example to show how to get the modified hamming distance between two tag sets T and T_q. Suppose $T=\{a, b, c, d\}$ and $T_q=\{a, e, f\}$. In this example, $(T-T_q)=\{b, c, d\}$ and $(T_q-T) = \{e, f\}$. All the possible pairs between the two unmatched tag sets are (b, e), (b, f), (c, e), (c, f), (d, e), and (d, f). Suppose that their related-degrees are 0.3, 0.4, 0.5, 0.2, 0.3, and 0.6 respectively. Accordingly, the first selected pair is (d, f) and the remained possible pairs are (b, e) and (c, e). Then the second selected pair is (c, e). The value of $SR(\{b, c, d\},\{e, f\})$ is estimated as 0.6+0.5=1.1. Besides, the modified hamming distance between $\{a, b, c, d\}$ and $\{a, e, f\}$ is computed as follows:

$mdist(\{a, b, c, d\}, \{a, b, c, d\})$

$= |\{a, b, c, d\}|+|\{a, e, f\}|-2(|\{a\}|+SR(\{a, b, c, d\},\{a, b, c, d\}))$

$= 4+3-2\times(1+1.1)) = 2.8$

B. Modified Estimation for Search Bounds

The modified hamming distance function always gets equivalent or smaller value than the hamming distance function for the same pair of tag sets. Therefore, we modify the estimating methods for 1st-UB(T_q, C) and 1st-LB(T_q, C) used in the searching algorithm in order to apply the bounding mechanism for the modified hamming distance. It can be proved that $SR((T-T_q),(T_q-T)) \leqslant \min(|T-T_q|, |T_q-T|)$.

1) Modified Search Bound Estimation for Clusters

According to the first level cluster bounding mechanism proposed in [6], the hamming distance between the query tag set T_q and each tag set T in a cluster C satisfies the following two non-equal equations:

$$diff(T_q, C.Co)+ diff(C.Ci, T_q) \le dist(T_q, T) \quad \text{............(6)}$$

$$dist(T_q, T) \le diff(T_q, C.Ci) + diff(C.Co, T_q) \quad \text{..........(7)}$$

The modified hamming distance between the query tag set T_q and each tag set T in cluster C, denoted as $mdist(T_q, T)$, is equivalent to $dist(T_q, T)$-$2SR((T,T_q)$. In other words, the modified hamming distance $mdist(T_q,T)$ satisfies the following two non-equal equations:

$$diff(T_q, C.Co)+ diff(C.Ci, T_q)-2SR((T,T_q) \le mdist(T_q,T) \quad \text{...(8)}$$

$$mdist(T_q, T) \le diff(T_q, C.Ci)+diff(C.Co, T_q)-2SR((T,T_q) \quad \text{...(9)}$$

According to the definition of $SR((T,T_q)$, it is guaranteed that $0 \le SR((T,T_q) \le min(|T_q - T|, |T - T_q|)$. In addition, $|T_q - T| \le |T_q - C.Ci|$ and $|T- T_q| \le |C.Co - T_q|$ because $C.Ci \subseteq T$ and $T \subseteq C.Co$. Consequently, it is derived that

$$0 \le SR((T,T_q) \le min(|T_q - C.Ci|, |C.Co - T_q|) \quad \text{............(10)}$$

Consequently, the upper bound and lower bound estimations for checking the modified hamming distance between the query tag set T_q and each tag set T in cluster C are as follows:

$$\text{1st-LB}'(T_q, C)= diff(T_q, C.Co) + diff(C.Ci, T_q) - min(|T_q - C.Ci|, |C.Co - T_q|) \times 2 \quad \text{..........(11)}$$

$$\text{1st-UB}'(T_q, C) = diff(T_q, C.Ci) + diff(C.Co, T_q) \quad \text{.......(12)}$$

2) Modified Search Bound Estimation for Batches

The second level cluster bounding mechanism proposed in [6] guarantees that the hamming distance between the query tag set T_q and each tag set T in a batch B of cluster C satisfies the following two non-equal equations:

$$(diff(T_q,C.Co) +|diff(C.Co, T_q)- dvo(C,B)|) \le dist(T_q, T) \quad \text{...(13)}$$

$$dist(T_q, T) \le (dist(T_q, T) + dvi(C,B)) \quad \text{.........................(14)}$$

Therefore, the modified hamming distance $mdist(T_q,T)$ satisfies the following two non-equal equations:

$$(diff(T_q,C.Co)+|diff(C.Co,T_q)-dvo(C,B)|)-2SR((T,T_q) \le mdist(T_q,T) \quad \text{...(15)}$$

$$mdist(T_q, T) \le (dist(T_q, T) + dvi(C,B)) -2SR((T,T_q) \quad \text{........(16)}$$

According to non-equal equation (10), the upper bound and lower bound estimations for checking the modified hamming distance between the query tag set T_q and each tag set T in a batch B of cluster C are as follows:

$$\text{2nd-LB}'(T_q, C, B)= diff(T_q,C.Co)+|diff(C.Co, T_q)-dvo(C,B)| - min(|T_q - C.Ci|, |C.Co - T_q|) \times 2 \quad \text{..........(17)}$$

$$\text{2nd-UB}'(T_q,C,B)=dist(T_q,C.Ci)+ dvi(C, B) \quad \text{................(18)}$$

V. Performance Evaluation

A systematic study is performed to evaluate the performance efficiency of the proposed Multi-level Hierarchical Index structure with Bounding mechanism (MHIB). In the experiments, the MHIB approach is compared with two related works: the Two-level Index structure with Bounding mechanism (TIB) proposed in [6] and the naïve method (Naïve) which sequentially checks all the tag sets to find answer of a query. All of these algorithms were implemented using Java in the Netbean platform. The experiments were performed on a 2.67GHz Intel Core2 Quad CPU with a DDR2 2G main memory and running Microsoft Windows XP.

We use a real tag dataset crawled from the Flickr social tagging system for the experiments. The data set is crawled by giving an initial query to retrieve the image ids in the first returned 10 pages of results as seeds. Then the tags in these images are used as queries recursively until we have collected 20,000 images and their tag sets. The dataset contains 60,230 unique tags and the average carnality of a tag set is 9.97.

In each experiment, one of the setting among the given threshold values at run time is varied to observe the changing trend of execution time. When performing the MHIB approach, the three threshold values are set as follows:

maxd_root is 50, *maxd_leaf* is 30, and the *maxd_batch* is 10. For the TIB approach, the *maxd_root* is set to 50. We randomly selected 100 tag sets from the dataset as test queries, where the average cardinality of the query tag sets is 10.5. For each experiment, the average execution time is obtained by performing the 100 different queries.

In the following, using the two difference distance measures: the hamming distance and the modified hamming distance, we observe the performance of similarity search, deletion and update operations respectively for the three approaches.

A. *Perfomed by Using Hamming Distance*

1) *Execustion Time of Performing Similarity Search*

a) *Varying the query threshold (δ)*

Fig.3(a) shows the execution time of performing similarity search for the three approaches by varying the query threshold δ from 0 to 20. The MHIB approach performs better efficiency for similarity searching than the other two approaches especially for the small δ setting. The reason is that we require the tag sets within a cluster must have at least one common tag. Therefore, the tag sets stored in a cluster got by the MHIB approach are more similar than the one got by the TIB approach. Besides, the MHIB approach is a forest like-structure, even we couldn't prune a cluster at the root level from the search space, it still has chance to prune the clusters at the next level from the search space. Accordingly, the pruning effect of the MHIB is better than the one of the TIB approach. If the pruning strategy according to the estimated distance bounds does not work well, the distance between each tag set and the query has to be calculated. It makes the TIB approach spend the additional cost for checking the distance bounds of clusters and batched but degrades to the naïve method. That is why the TIB approach performs worse than the naïve method in this case. Because both the average cardinality of a query and a tag set is about 10, it is reasonable to set the value of δ less than or equal to 10. In the case when δ is set to be 10, the returned tag sets are having about 50% overlap with the query on their tags.

b) *Varying the maxd_root*

In this experiment, the setting of *maxd_root* is varied from 10 to 50 to observe the execution time of similarity search with δ is 0 and 10, which implies the exactly match and a lower similarity requirement of a query, respectively. The results of the two different setting of δ are shown in Figure 3(b) and 15(c), respectively.

When $\delta = 0$, as the result shown in Figure 3(b), the MHIB approach performs much more efficiently than the other two methods. It indicates that the inverted list of the clusters at the root level provides very good effect for reducing the search space of clusters for the MHIB approach.

Figure 3(c), shows the execution times of the three approaches when $\delta = 10$. When the *maxd_root* threshold is set to a larger value, it means that the tag sets assigned in the same cluster are less similar to each other than the ones in a cluster constructed with a smaller *maxd_root*. Consequently, fewer clusters will be formed by both the MHIB and the TIB approaches with a larger value of *maxd_root*. That is why the execution times of both both the MHIB and the TIB approaches decrease as the value of *maxd_root* increases. When the value of the *maxd_root* threshold is as large as 50, the pruning strategy of the TIB approach becomes less effective because too many dissimilar tag sets are assign to a cluster. That is why the execution time of the TIB approach with *maxd_root*=50 is more than the one with *maxd_root*=40. However, for the MHIB approach, more than one level of clusters will be constructed when the cluster at the root level violates the *maxd_leaf* constraint. Therefore, the MHIB approach still has good pruning effect with *maxd_root*=50.

c) *Varying the cardinality of the query tag set*

In this experiment, the cardinality of the query tag set, called query length in the following, is varied from 4 to 20. The execution times of similarity search by using the three approaches with δ is 0 and 10 are shown in Figure 3(d) and 15(e), respectively.

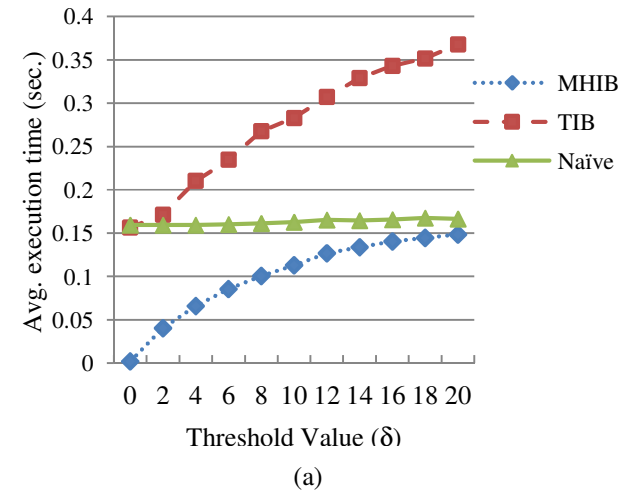
Without surprising, the execution times of all the three approaches increase as the query length increases when $\delta = 0$. The reason is that the larger the query length, the more time needed to perform matching of the members in the tag sets. When $\delta = 10$, most clusters in the index structure could not be pruned from the search space when the query length is less that the query threshold (i.e. 4 and 8 in this case). However, the execution times of both the MHIB and the TIB approach decrease as the query length increases from 8 to 12. The reason is that the pruning effect of the bounding mechanisms starts to work well when the query length is greater than δ. After that, the execution times of the two approaches keep increasing slightly as the query length increases.

2) *Execustion Time of Updating the Index Structure*

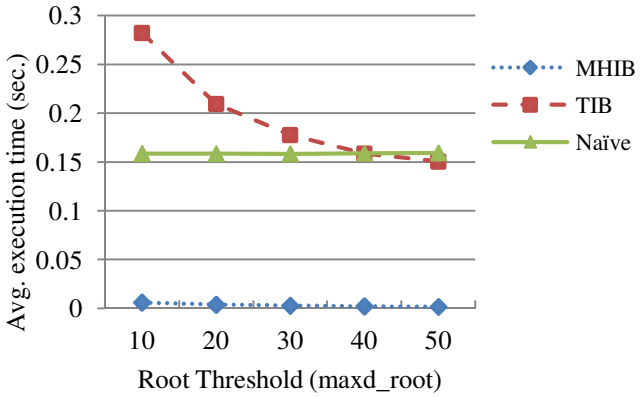
We also compare the execution times of the three approaches when performing deletion or updating operations on the index structure.

As the result shown in Figure 3(f), the MHIB approach performs much more efficiently than the other methods when performing the deletion operation. The reason is that the inverted list is used to select the clusters that might contain the tag set to be deleted, which effectively reduces the number of clusters need to be checked. The naïve method performed poorly because it has to check all the tag sets one by one.

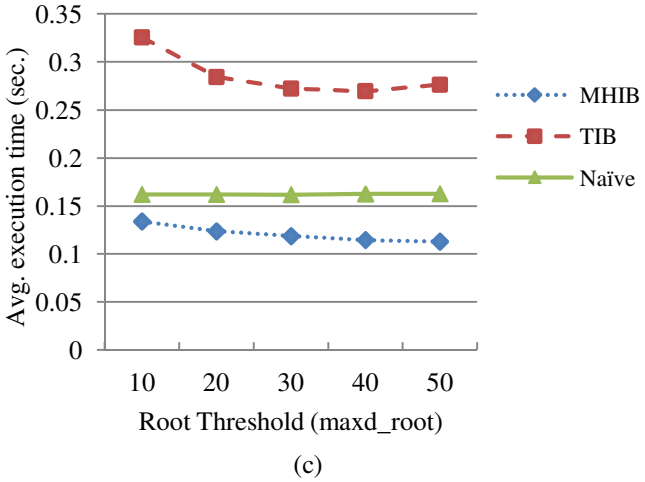
The MHIB approach has to find the best cluster among the clusters for inserting a new tag set. Therefore, it requires more time to perform an insertion operation. However, the MHIB approach performs a deletion operation much faster than the other two approaches. Consequently, as shown in Figure 3(g), the MHIB approach also has the best performance on the execution time of update operation.



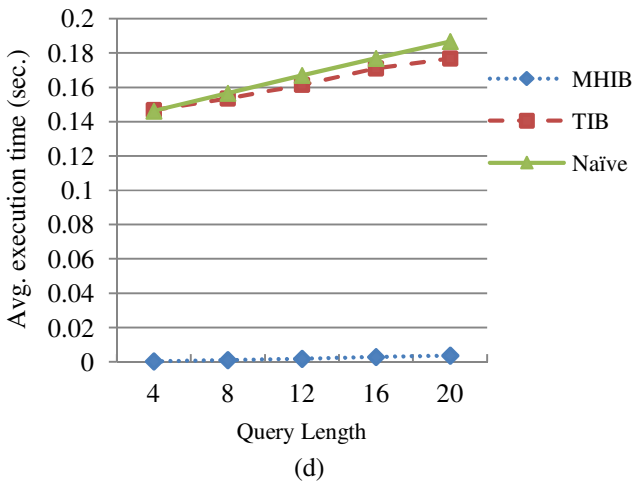
(a)



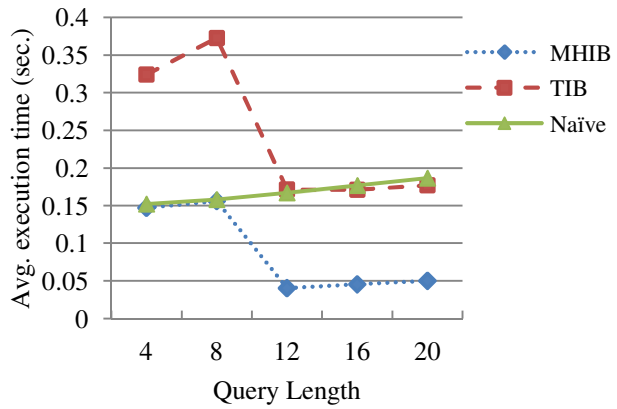
(b)



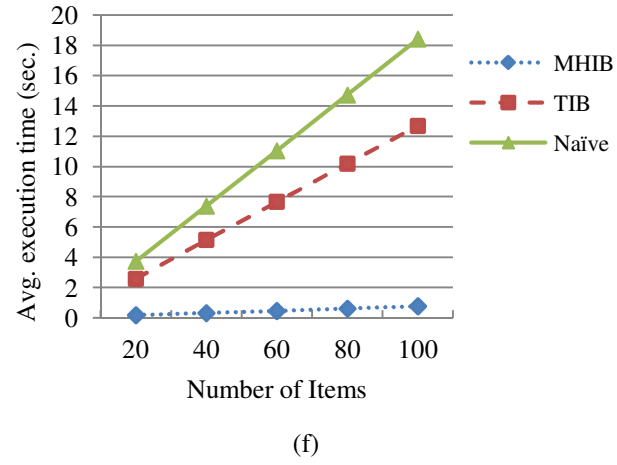
(c)



(d)



(e)



(f)

Figure 3. The results of performance evaluation.

B. Perfomed by Using Modified Hamming Distance

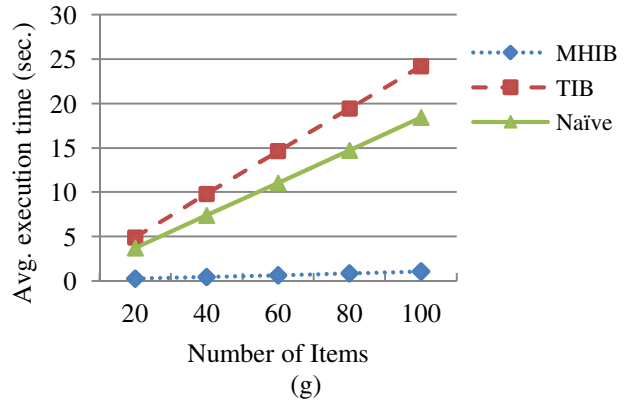
The modified hamming distance mainly influence the performance of similarity search because it requires higher cost for computing the distance between a tag set and the query tag set than the hamming distance. Accordingly, the execution time of performing a similarity search is significantly influenced by the pruning effectiveness of the proposed index structure. In this part of experiments, we evaluate the execution times of the MHIB and the TIB approaches for performing similarity search operations by varying the setting of query threshold δ, *maxd_root*, and query length, respectively.

a) Varying the query threshold δ

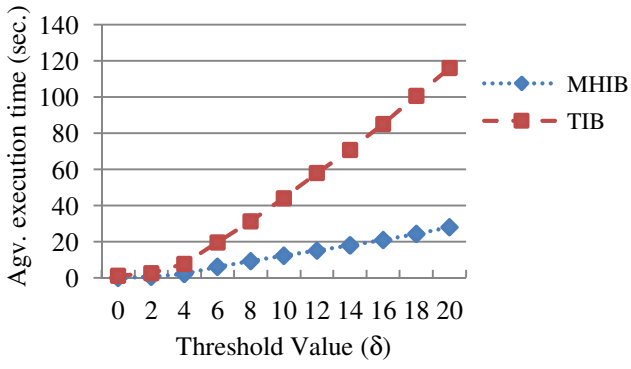
Figure 3(h) shows the execution times of the two approaches by varying the query threshold δ. When the value of δ increases, more tag sets satisfy the query. It clauses the number of clusters which can be pruned from the search space decreases. That is why the execution time of both approaches increases as δ increases, which is consistent with the results shown in Figure 3(a). However, the modified hamming distance between a tag set and a query tag set is smaller than the hamming distance between them. Accordingly, for the same query threshold setting, using the modified hamming distance to perform similarity measures on tag sets gets more tag sets as the answers than using the hamming distance. Besides, the cost of computing a modified hamming distance between two tag sets is higher than the one of computing their modified hamming distance. Consequently, the increasing rate of the execution time for the TIB approach shown in Figure 3(h) is faster than the one shown in Figure 3(a). However, for the MHIB approach, the effect on the execution time is less significant by varying the query threshold. It indicates that the MHIB approach is well scalable with respect to δ.

b) Varying the maxd_root

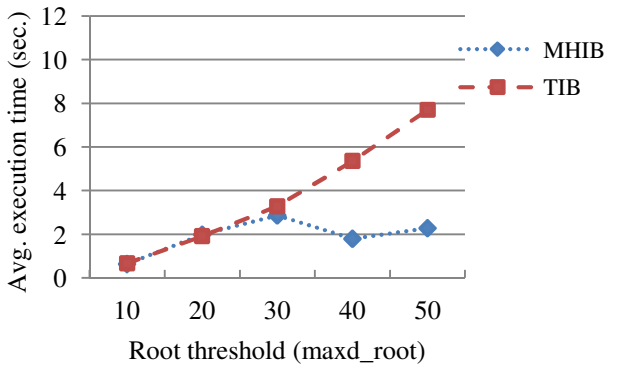
The execution times of the two approaches by varying the *maxd_root* are shown in Figure 3(i), where the query threshold δ is set to 4. When the value of the *maxd_root* increases from 10 to 30, the execution times of both methods increase linearly. During this setting range, the constructed index structure of the MHIB approach only has one level of clusters, which is similar to the TIB approach. However, the similarity of a cluster formed by the MHIB approach is higher than the one formed by the TIB approach. For this reason, the MHIB approach performed similarity searches more efficiently than the TIB approach. When the *maxd_root* increases from 30 to 50, more than one level of clusters is constructed by the MHIB approach, which performs better pruning effect than the one level of clusters. That is why the execution time of the MHIB approach decreases from 30 to 40 and the increasing rate starting from *maxd_root* = 40 is less than the one when *maxd_root* is less than 30.



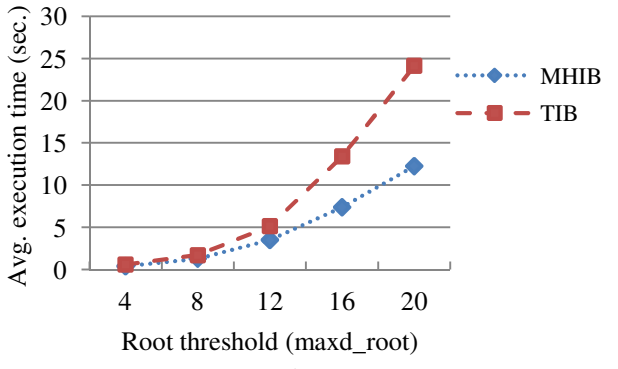
(g)



(h)



(i)



(j)

Figure 3. (cont.)

c) Varying the query length

When δ is set to 4, the execution times of the two approaches by varying the query length are shown in Figure 3(j). When the query length increases, the computation cost of performing a modified hamming distance increases exponentially. That is why the execution times of both approaches increase as the query length increases. Besides, for each approach, the increasing rate of the execution time by using the modified hamming distance is more significant the one by using the hamming distance as shown in Figure 3(c). However, because of a better pruning effectiveness of the constructed index structure, the scalability of the MHIB approach with respect to the query length is still better than the TIB approach when using the modified hamming distance for similarity measure.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, for providing efficient similarity search of tag set in a social tagging system, we propose a multi-level hierarchical index structure to group similar tag sets. Not only the algorithms of similarity searches of tag sets, but also the algorithms of deletion and updating of tag sets by using the constructed index structure are provided. Furthermore, we define a modified hamming distance function on tag sets, which consider the semantically relatedness when comparing the members for evaluating the similarity of two tag sets. This function is more applicable to evaluate the similarity search of two tag sets. Finally, a systematic performance study is performed to verify the effectiveness and the efficiency of the proposed strategies. By varying the query threshold, *maxd_root*, and query length, respectively, the results show that the proposed MHIB approach further improves the pruning effect of the TIB approach. Especially, the MHIB approach is well scalable with respect to the three parameters when using either the hamming distance or the modified hamming distance for similarity measure. Although the insertion operation of the MHIB approach requires higher cost than the naïve method, with the assistant of the constructed inverted list of clusters, it performs faster than the TIB approach. Besides, the cost of performing deletion operation by using the MHIB approach is much less than the other two approaches and so is the update operation.

In the literature, there are other distance functions for evaluating the distance or similarity between two sets. We will explore different measuring functions of sets to find the most effective measuring function for the similarity search of tag dataset and modify the proposed index structure to support the corresponding pruning strategy.

References

[1] C. Aggarwal, J.L. Wolf, and P.S. Yu, "A New Method for Similarity Indexing of Market Basket Data," in Proceedings of the ACM International Conference on Management of Data (SIGMOD), 1999.

[2] P. Agrawal, A. Arasu and R. Kaushik, "On Indexing Error-Tolerant Set Containment," in Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2010.

[3] A. Arasu, V. Ganti, and R. Kaushik, "Efficient Exact Set-Similarity Joins," in Proceedings of the 32nd International Conference on Very Large Databases (VLDB), 2006.

[4] R.J. Bayardo, Y. Ma, and R. Srikant, "Scaling Up All Pairs Similarity Search," in Proceedings of the 16th International Conference on World Wide Web (WWW), 2007.

[5] A. Budura, S. Michel, P. Cudre-Mauroux, and K. Aberer, "Neighborhood-based Tag Prediction," in Proceedings of the 6th Annual European Semantic Web Conference (ESWC), 2009.

[6] J.C. Chuang, C.W. Cho, and A.L.P. Chen, "Similarity Search in Transaction Databases with a Two Level Bounding Mechanism," in Proceeding of the 11th International Conference of Database Systems for Advanced Applications (DASFAA), 2006.

[7] D. Carmel, H. Roitman, and E. Yom-Tov, "Social bookmark weighting for search and recommendation," in Proceedings of the 19th International Journal on Very Large Data Bases (VLDB), 2010.

[8] Q. Jing and Y.Rui, "Localized Signature Table: Fast Similarity Search on Transaction Data," in Proceeding of the 13th ACM International Conference on Information and knowledge Management (CIKM), 2004.

[9] N. Mamoulis, D. Cheung, and W. Lian, "Similarity Search in Sets and Categorical Data Using the Signature Tree," in Proceedings of 19th International Conference on Data Engineering (ICDE), 2003.

[10] G.S. Manku, A. Jain, and A.D. Sarma, "Detecting NearDuplicates for Web Crawling," in Proceedings of the 16th ACM International conference on World Wide Web (WWW), 2007.

[11] A. Nanopoulos and Y. Mamolopoulos, "Efficient Similarity Search for Market Basket Data," The VLDB Journal — The International Journal on Very Large Data Bases, Vol. 11 Issue 2, October 2002.

[12] C. Ordonez, E. Omiecinski, and N. Ezquerra, "A fast Algorithm to Cluster High Dimensional Basket Data," in Proceedings in Proceedings of 17th International Conference on Data Engineering (IEEE), 2001.

[13] E. Tousidou, A. Nanopoulos, and Y. Manolopoulos, "Improved Methods for Signature-Tree Construction," in The Computer Journal, Vol. 43, No. 4, 2000.

[14] D. Vandic, J.W. Dam, F. Hogenboom, "A Semantic Clustering-Based Approach for Searching and Browsing Tag Spaces," in Proceedings of the ACM 26th Symposium On Applied Computing (SAC), 2011.

[15] C. Xiao, W. Wang, and X. Lin, "Ed-Join: An Efficient Algorithm for Similarity Joins With Edit Distance Constraints," in Proceeding of the 17th International Journal on Very Large Data Bases (VLDB), 2008.

[16] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k Set similarity Joins," in Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2009.

[17] C. Xiao, W. Wang, X. Lin, and X. Yu, "Efficient Similarity Joins for Near Duplicate Detection," in Proceedings of the 17th International Conference on World Wide Web(WWW), 2008.

[18] C. Yeung, N. Gibbins, and N. Shadbolt, "User-induced Links in Collaborative Tagging Systems," in Proceeding of the 18th ACM International Conference on Information and Knowledge Management(CIKM), 2009.