Efficient Search for Data with Numerical and Categorical Attributes Based on Dual Locality-Sensitive Hashing

Kyung Mi Lee and Keon Myung Lee*  
Department of Computer Science, Chungbuk National University, Chungdae-ro 1, Seowon-ku, Cheongju, 28644, Korea; Kmlee07@cbnu.ac.kr, kmlee@cbnu.ac.kr

Abstract

Background/Objectives: Similarity search is a fundamental task in many domains. For large volumes of data, it is sometimes preferable to obtain approximate results instead of pursuing exact results that require long computation times. Methods/Statistical Analysis: The proposed method expresses a query using a fuzzy ball for the numerical data space and a conjunction of ground or general terms for categorical domains. It exploits a dual locality-sensitive hashing technique that constructs separate locality-sensitive hashing tables for the numerical and categorical data spaces. It determines buckets for a query by considering the relative distances of the numerical part of the query to the subspace boundaries and using concept hierarchies for the categorical attributes. It may also use a Bloom filter to select the candidate data set to be examined from among the determined buckets. Findings: The proposed approximate search technique was applied to two data sets. The portions of data to be examined were 0.02712% and 0.00084% for the first and the second data set. The average numbers of numerical buckets examined were 1.42 and 2.53 for the data sets, respectively. The figures mean that the proposed method significantly reduces the number of candidates to be considered and hence saves greatly computation cost. In addition, the proposed method is unique that it can be applied to data set with both numerical and categorical attributes. Improvements/Applications: The proposed locality-sensitive hashing method can be applied to approximate search tasks for large volume of data containing both numerical and categorical attributes.

Keywords: Locality-Sensitive Hashing, Nearest Neighbor Search, Similarity Search, Space Partitioning

1. Introduction

Similarity search problems are prevalent across a number of domains, including information retrieval, content-based image retrieval, computer vision, bioinformatics, and computational linguistics. Similarity computation between a data object and a query is usually simple, but it becomes burdensome when the volume of data is large. Several indexing and hashing techniques have been proposed in response to computation time concerns. The indexing techniques organize the data set into a structure, usually tree-based, in order to minimize the number of computations for locating specific data. They are efficient for relatively low-dimensional spaces, but they usually deteriorate to almost an exhaustive search for high-dimensional spaces. Hashing is a technique to determine the location of data with a simple computation, although there are chances to have collisions which require extra computations. When the data set is large and its dimension is high, the traditional indexing and hashing techniques lose their effectiveness. Moreover, when similarity-based or approximate search is needed, indexing or hashing techniques that support nearest neighbor search are required.

Similarity search is difficult for data having both numerical and categorical attributes because there are no widely accepted similarity or distance measures for them as yet. We may encounter large volumes of such data in many application domains. To address the similarity search problem for such data sets, we propose a locality-sensitive hashing-based search method.

The proposed method expresses a query with a fuzzy
ball for numerical space and a conjunction of ground or general terms for categorical domains. It exploits a dual locality-sensitive hashing technique that constructs separate locality-sensitive hashing tables for the numerical and categorical spaces. It determines the buckets for a query by considering the relative distances of the numerical part of the query to the subspace boundaries and using concept hierarchies for the categorical attributes. When the number of data objects to be considered is large, it may use a Bloom filter\(^{10}\) to select the candidate data set to be examined from among the buckets.

The remainder of the paper is organized as follows: Section 2 presents related work, and Section 3 introduces the proposed search method for large volumes of data in high-dimensional data space. Section 4 shows some experimental results of the proposed method, and, finally, Section 5 draws conclusions.

## 2. Related Work

### 2.1 Similarity Search

The similarity search problem is closely related to the nearest neighbor search problem, which seeks the data point closest to a query. In the database literature, a point query \(P(q)\) is used to find the data point \(x\) that meets its conditions from the data set \(X\).

\[
P(q) = \{x \mid d(q,x) = 0 \text{ for } x \in X\}, \tag{1}
\]

where \(d(q,x)\) indicates the distance between \(q\) and \(x\). A range query \(RQ(q,r)\) specifies the data objects that are located within a distance \(r\) from a query point \(q\):

\[
RQ(q,r) = \{x \mid d(q,x) \leq r \text{ for } x \in X\}. \tag{2}
\]

As another alternative, the nearest neighbor \(NN(q)\) search for a query point \(q\) seeks the data point closest to the query without requiring an exact match or specifying an exact distance:

\[
NN(q) = \{x \mid d(q,x) \leq d(q,y) \text{ for all } y \in X - \{x\}\}. \tag{3}
\]

In the \(k\)-nearest neighbor search problem, the top \(k\) nearest data points \(kNN(q,k)\) are searched with respect to a query point \(q\):

\[
kNN(q,k) = \{x \in A \mid A \subseteq X, |A| = k, d(q,x) \leq d(q,y) \text{ for all } y \in X - A\}. \tag{4}
\]

While the range query finds all data points located within the given distance \(r\) from the query point, the \(k\)-nearest neighbor search is concerned with finding only the top \(k\) nearest ones.

Fuzzy queries are specified using membership functions\(^5\), which express qualitative or linguistic terms in a quantitative manner. One form of fuzzy query is shown by Eq. (5), in which \(\mu_q(x)\) denotes a membership function to measure the extent to which data point \(x\) satisfies the constraint imposed by query \(q\) and the minimum satisfaction degree \(m\).

\[
FQ(q,m) = \{\{x : \mu_q(x) \} \mid \mu_q(x) \geq m \text{ for } x \in X\} \tag{5}
\]

Various indexing structures have been developed to support similarity and nearest neighbor search\(^{1,11-18}\). Most of them are based on the partitioning principle, according to which the search space is divided into subspaces and a search is performed for a similar or nearest point starting from the subspace in which the query point is located. Several tree-indexing structures have been developed, in which the space is partitioned using either spherical balls or hyperplanes. Ball partitioning methods include vantage point tree, fixed-queries tree, multi-vantage point tree, and \(m\)-tree methods, among others\(^{1,3,4}\). Some examples of generalized hyperplane partitioning methods are \(kd\)-tree, generalized hyperplane tree, spatial tree, Principal Component Analysis (PCA) tree, and random projection tree\(^{1,3,4}\).

When we enter a high-dimensional space, we suffer from the curse of dimensionality\(^{19-21}\). This implies that as the dimensionality of data space increases, the distances among all data objects become more similar. While tree-structured indexing techniques work quite well in lower-dimensional space, in high-dimensional space they do not always do better than a brute-force linear search. To deal with high-dimensional data spaces in similarity searches, hash-based techniques called locality-sensitive hashing have been developed\(^{9,11}\).

### 2.2 Locality-Sensitive Hashing

Hash-based indexing techniques usually focus on mapping an even distribution of keys over the buckets, and the major concern is to quickly locate data given their keys\(^5\). A new class of hash techniques called Locality-Sensitive Hashing (LSH) has been developed for approximate searches. Locality-sensitive hash techniques implement hash functions that map similar keys to the same bucket and different keys to different buckets with high probability.

The notion of locality-sensitive hashing was first proposed by Indyk and Motwani\(^9\) and is defined as follows. For any two data objects \(p, q \in S\), a set \(U\) of buckets, functions \(h \in H\), a distance function \(d(p,q)\), probability function \(P_0\), and non-negative constants \(r_1, r_2 \in [0,\infty)\) \((r_1 < r_2)\) and \(P_{r_1}, P_{r_2} \in (0,1)\) such that \(P_{r_1} > P_{r_2}\), an
LSH family $H = \{ h \mid S \rightarrow U \}$ is a family of functions such that the following conditions are satisfied:

If $d(p,q) \leq r_1$, then $P_{r_1}(h(p) = h(q)) \geq P_r$, and if $d(p,q) \geq r_2$, then

$$P_{r_2}(h(p) = h(q)) \leq P_r.$$   \hspace{1cm} (6)

The functions satisfying the above properties are called $(r_1, r_2, P_1, P_2)$-sensitive\textsuperscript{21}. Most LSH methods encode data into short binary strings for hash codes; data objects with the same hash code have a high probability of being similar to each other. Various LSH methods have been developed for approximate similarity search\textsuperscript{9,11,12,17-19}. Most of them use multiple functions to construct hash codes, each function producing a single bit that is later concatenated with other bits to form a binary string.

A wide array of LSH methods have been developed for numerical data sets\textsuperscript{16-18}. The projection-based method is an LSH method that uses several randomly selected projection vectors that define hyperplanes\textsuperscript{5}, each of which takes the role of a hash function, producing a binary value. The boost similarity-sensitive hashing method uses a machine learning–based LSH technique that applies AdaBoost\textsuperscript{20} to both the sampled similar pairs and the dissimilar pairs of data\textsuperscript{15}. The Restricted Boltzmann Machine (RBM)-based LSH method trains a stacked RBM with multiple layers, the upper layers gradually having smaller numbers of nodes\textsuperscript{25}. The density-sensitive hashing method chooses the separating hyperplanes by first finding the clusters using the k-means algorithm\textsuperscript{21} and then searching for the median planes to separate adjacent groups in such a way as to produce the highest entropy score\textsuperscript{7}. These numerical LHS methods have been proved to be practical and successful in various similarity search applications such as image retrieval\textsuperscript{18}.

Although many data sets have categorical attributes, these do not usually provide any natural information regarding distances between values. Lee and Lee\textsuperscript{22} proposed an LSH method for categorical data that uses data-driven distance measures to create a similarity matrix for the attribute values and then groups the categorical values into disjoint groups using hierarchical clustering based on the similarity matrix\textsuperscript{23,24}. After that, mapping functions are defined to project the attribute values into disjoint group identifiers; these are combined to build an LSH function that encompasses all the categorical attributes. Although the categorical LSH method is a meaningful approach, it is restrictive in that the hash codes can be rather long when there are many categorical attributes each of which produces its own hash bits.

Many data sets contain both numerical and categorical attributes simultaneously. To deal with such data in an LSH manner, the dual hash technique was proposed, which uses both numerical LSH and categorical LSH techniques for the corresponding attributes\textsuperscript{21}. It maintains two sets of hash tables, one for numerical attributes and the other for categorical ones.

### 2.3 Bloom Filter

The Bloom filter is a technique to filter streaming data so that elements belonging to a particular key set $S$ pass through and others are ignored\textsuperscript{5}. It consists of an array $B$ of $n$ bits, initially set to 0, and a set of hash functions $h_1, h_2, \ldots, h_k$. The range of each hash function is from 1 to $n$, corresponding to the bit position of the array. To add a key $K$ to $S$, the filter applies it to $k$ hash functions to get $k$ array positions and sets the bits at all these positions to 1.

$$B[h_i(K)] = 1 \text{ for } i = 1, \ldots, k.$$   \hspace{1cm} (7)

To check whether a key $K$ is in $S$, the algorithm determines the array positions by applying the key to the hash functions, and it judges that the key is in $S$ only when all bits at the corresponding positions are 1.

A Bloom filter never fails to detect data having keys that belong to the key set $S$; however, there is a chance that incorrect data will also be passed. The probability of a false positive is $(1 - e^{-km/n})^k$, where the size of the key set $S$ is $m$, the array has $n$ bits, and $k$ hash functions are used.

### 2.4 Fuzzy Theory

Fuzzy sets and their degrees of membership provide a useful tool for expressing vague or uncertain collections and quantities in numeric form\textsuperscript{25}. In particular, they allow us to convert qualitative linguistic terms such as “small” or “about 12” into quantitative values.

When membership functions are defined on numerical domains, they are usually specified using special unimodal forms like triangular, trapezoidal, or Gaussian fuzzy numbers\textsuperscript{5}. While these fuzzy numbers are easily defined on a single numerical domain, we are concerned with multi- or high-dimensional numerical domains. To help deal with fuzzily quantified numerical subspaces, we define the notion of a fuzzy ball as follows: A fuzzy ball $FB(c, \mu)$ is a ball-shaped membership function centered at point $c$, at which point the membership degree is the largest and gradually decreases as it goes away from $c$. Figure 1 shows a fuzzy ball and its membership function defined with respect to the distance from its center.
Concept hierarchies are sometimes defined for categorical attributes to describe the conceptual hierarchy among the categorical terms, where parent nodes indicate terms of greater generality than that of their children. When the terms represent categories, some terms may have partial membership with multiple general terms. Such concept hierarchies can be extended to fuzzy conceptual hierarchies, in which the edges are weighted with the membership degree with which the child term belongs to its parent term. Figure 2 shows a fuzzy concept hierarchy regarding software.

Figure 2. Fuzzy concept hierarchy\textsuperscript{25}. Edge weights denote the degrees of membership to parent terms.

3. The Proposed Method

The overall architecture of the proposed method is shown in Figure 3. The data set to be searched is assumed to contain both numerical and categorical attributes. For flexible query representation, the method supports fuzzy queries by incorporating fuzzy balls and generalized categorical terms along with the threshold for the degree of matching. For the efficient handling of large volumes of numerical and categorical data, a dual LSH technique is employed, with both numerical LSH and categorical LSH modules. The numerical LSH module hashes data into its own buckets based on the numerical part of the data, and the categorical LHS module likewise maintains its own buckets with respect to the categorical part of the data.

To support fuzzy query and efficient bucket management, the proposed method develops an improved bucket selection strategy for numerical buckets and introduces a new categorical LSH technique. For a given fuzzy query, the LSH functions determine the buckets from both the numerical and categorical bucket sets in which similar data are to be searched. The intersection of the data in the selected numerical buckets with those in the selected categorical buckets comprises the candidate data set to be examined. To reduce execution time when the volume of data is large, a Bloom filter can optionally be used to perform the intersection. The candidate set is now examined to compute the degree to which each data object satisfies the fuzzy query. Only those data objects having a degree of satisfaction greater than the threshold are returned as the query result.

Figure 3. Overall architecture of the proposed method. BN\textsubscript{1}, \ldots, BN\textsubscript{n} denote the buckets for the numerical LSH, and BC\textsubscript{1}, \ldots, BC\textsubscript{m} indicate the buckets for the categorical LSH.
3.1 Fuzzy Query Expression

To deal with high-dimensional numerical attributes, fuzzy balls are allowed as fuzzy constraints, which are defined as a unimodal membership function centered at a point in the numerical attribute space. The constraints on categorical attributes are selectively enforced; hence, the user might not care about the values some attributes have. A Fuzzy Query FQ for similarity search is described by specifying a fuzzy ball on the numerical attribute space and fuzzy constraints on some categorical attributes, as follows:

\[(NS = FB, C_{i_{1}} = t_{1}, ..., C_{i_{r}} = t_{r}, Th = \theta) \quad \text{where } FB = (c = (l_{1}, ..., l_{d}), \text{proximity} = LT)\]. (8)

Here \((l_{1}, ..., l_{d})\) denotes the center of the fuzzy ball, \(LT\) is a linguistic term expressing the proximity, \(\theta\) is the threshold for the degree of satisfaction necessary for a match, and \(C_{i_{k}} = t_{k}\) represents the condition that categorical attribute \(C_{i_{r}}\) should be comparable with term \(t_{k}\), which can be either a ground term or a general term. The ground terms refer to the categorical values that appear in the data set. In a query, not all categorical attributes need to have a condition statement.

3.2 Locality-Sensitive Hashing for Categorical Attributes with Selective Query Support

Categorical hashing for categorical attributes is conducted using the following strategy: If a (fuzzy) concept hierarchy is given for an attribute, it is used to determine its constituent code for the hash function. Otherwise, the hierarchical tree for the attribute is constructed using a data-driven distance matrix as in the method of Lee and Lee [26]. To balance the bucket size, a randomly sampled subset of the data set is used. For the samples, we count the occurrences of distinct categorical values for each attribute. In the concept hierarchies, a concept node may have multiple parent nodes. Once the value occurrence counts are obtained, the leaf nodes are tagged with the corresponding counts. The occurrence count of each internal node is the sum of the occurrence counts of the internal node's child nodes. The categorical LSH method partitions the value set of categorical attributes into the number of partitions given by its pre-assigned resolution. For this purpose, we sort the nodes in increasing order of their occurrence count and compute the expected average count by dividing the number of samples by the attribute resolution. Choosing partitions in a greedy manner, we start with the node having the count closest to the expected average and continue choosing nodes one by one so that the set of the chosen partitions covers the whole set of leaf nodes in a balanced manner.

The following procedure, \(\text{Partition-Cat-Space}(S, C, r, TR)\), partitions the categorical value set of a categorical attribute \(C\) into the set \(P\) of \(r\) subsets using sample data set \(S\) and concept hierarchy \(TR\):

\[\text{procedure Partition-Cat-Space}(S, C, r, TR)\]

\[\text{input: sample data set } S, \text{ attribute } C, \text{ resolution } r, \text{ concept hierarchy } TR\]

\[\text{output: a set } P \text{ of categorical value subsets}\]

\[\begin{align*}
\text{begin} \\
\quad \text{count} = 0 \\
\quad \text{for } e \in VS(C) \\
\quad \quad o(C, e, S) = \text{Count-Occurrence}(C, e, S) \\
\quad \quad \text{count} = \text{count} + o(C, e, S) \\
\quad \quad oc(nn(e)) = o(C, e, S) \\
\quad \quad ac = \text{count}/r \\
\quad \text{Propagate-Occurrence-Count}(TR) \\
\quad P = \emptyset \\
\quad m = 0
\end{align*}\]
Cs = VS(C)
while (m < r and Cs ≠ ∅)
    nd = argmin_{n in S} \{|oc(n) - ac|\}
    P = P \cup \{tn(nd)\}
for ind ∈ tn(nd)
    for pnd ∈ an(nd)
        oc(pnd) = oc(pnd) - oc(ind)
    Cs = Cs - tn(nd)
    m = m + 1
while (m < r)
    R = argmax_{E \in E} \{Σ_{e \in E} oc(n)\}
    P = P - R
    R, R' = Split-Even(R, TR)
    P = P \cup \{R_1\} \cup \{R_2\}
if (CS ≠ ∅)
    R = argmin_{E \in E} \{Σ_{e \in E} oc(n)\}
    P = P - R
    R = R \cup \{Cs\}
end.

In the above procedure, Count-Occurrence(C, e, S) is the procedure to count the occurrences of value e for attribute C in data set S. Propagate-Occurrence-Count(TR) propagates occurrence counts upward from leaf nodes in a concept hierarchy whose leaf nodes are tagged with their occurrence counts.

procedure Propagate-Occurrence-Count(TR)
input: concept hierarchy TR with leaf nodes having their occurrence counts
output: concept hierarchy TR with all nodes having their occurrence counts
begin
    for nd ∈ IN(TR)
        oc(nd) = 0
    Q = IN
    while Q ≠ ∅
        R = \{nd | oc(cn) ≠ 0 for all cn ∈ ch(nd)\}
        for nd ∈ R
            M = \bigcup_{cn ∈ ch(nd)} tn(cn)
            oc(nd) = Σ_{cn ∈ M} oc(cn)
        Q = Q - R
    end.

Split-Even(R, TR) splits the set R into two subsets R_1 and R_2 for which the sums of their occurrence counts are approximately balanced.

procedure Split-Even(R, TR)
input: a set R of nodes, concept hierarchy TR
output: two subsets R_1, R_2 such that R = R_1 \cup R_2
begin
    ad = Least-Common-Ancestor(R, TR)
    STR(SV, SE) = Get-Sub-Hierarchy(TR, ad, R)
    bc = Σ_{nd \in STR-CN} oc(nd)/2
    nd = argmin_{nd \in STR-CN} \{|oc(nd) - bc|\}
    R_1 = ch(nd)
    R_2 = R - R_1
end.

Least-Common-Ancestor(R, TR) is the procedure that finds a least common ancestor of nodes in R of the concept hierarchy TR.

procedure Least-Common-Ancestor(R, TR)
input: a set R of leaf nodes, concept hierarchy TR
output: a least common ancestor node CAN
begin
    Q = Create-Queue()
    HM = Create-HashMap()
    for tn ∈ R
        h = 0
        PN = pa(tn)
        for nd ∈ PN
            Enqueue(Q, (nd, h+1))
        while !Empty-Queue(Q)
            (nd, ht) = Dequeue(Q)
            H(nd) = ht
            (hnd, cv) = Find(HM, nd)
            if hnd == null
                Insert(HM, (hnd, 1))
            else
                Update(HM, (hnd, cv+1))
        PN = pa(tn)
        for pd ∈ PN
            Enqueue(Q, (pd, ht+1))
    CS = \{dn | nd ∈ HM, cv = |R|\}
    CSH = argmin_{nd \in HM} \{H(nd)\}
    CAN = argmin_{nd \in CSH} \{oc(nd)\}
end.

Create-Queue(), Enqueue(), Dequeue(), and Empty-Queue() are operations for the data structure queue. Create-HashMap(), Insert(), and Update() are operations for a hash map that keeps records of key and value pairs. Find(HM, nd) retrieves the pair of the node and its value that has nd as a key from the hash map HM.

Get-Sub-Hierarchy(TR, ad, R) produces a subgraph STR(SV, SE) of TR, which contains all the nodes and edges from node ad to leaf nodes in R as follows:

SV = ad \cup de(ad)
SE = \{(na, nb) | na, nb ∈ SV, (na, nb) ∈ E\}
Once a categorical space is partitioned by Partition-Cat-
Space, a binary code is assigned to each partition using Assign-Binary-Codes. Assign-Binary-Codes(P, r) takes as input r sets of nodes, each of which is a partition, and assigns a unique binary code of length \( \lceil \log_2 r \rceil \) to each partition; hence, terms belonging to the same partition have the same binary code.

**procedure Assign-Binary-Codes(P, r)**

**input:** a partition set P of nodes, resolution r

**output:** a code table CT = \{ (v, code) \}

**begin**

\[ \text{len} = \lceil \log_2 r \rceil \]

\[ CT = \emptyset \]

\[ \text{code} = \text{binary string for value 0 of length len} \]

**for** vS ∈ P

**for** nd ∈ vS

\[ \text{CT} = \text{CT} \cup \{ (\text{lb}(\text{nd}), \text{code}) \} \]

**end.**

**end.**

Figure 4 shows how categorical attribute values are partitioned and how unique binary codes are assigned to ground terms. In the figure, the sample size is 48 and the resolution is set to 4; hence, the average count in a partition is 12. The partitions are therefore formed by Partition-Cat-Space to have about 12 counts. The partitions thus determined are \{g1, g2\}, \{g6, g7\}, \{g5, g7\}, and \{g5\}. Each partition is then assigned a unique binary code of length 2 by the procedure Assign-Binary-Codes.

![Figure 4](image)

**Figure 4.** Partitioning of categorical attribute values with ground terms g1, …, g7 and general terms i8, …, i12. The numbers in parentheses indicate occurrence counts for the ground terms and accumulated counts for the general terms. When resolution r is set to 4, the procedure Partition-Cat-Space comes to find partitions \{g1, g2\}, \{g6, g7\}, \{g5, g7\}, and \{g5\} sequentially. The procedure Assign-Binary-Codes can then give binary codes to the ground terms \{(g1,00), (g2,00), (g6,01), (g7,01), (g5,10), (g7,10), (g3,11)\}

Once the code tables are obtained for all categorical attributes, the binary map CCD(Di) for the categorical attribute part CV(Di) for data object Di is created using the next procedure, Code-for-Categorical-Domain(CTS, Di):

**procedure Code-for-Categorical-Domain(CTS, Di)**

**input:** code table set CTS = \{ CTv, …, CTq \}, data object Di

**output:** binary map CCD

**begin**

\[ CCD = \text{null} \]

**for** i = 1 to q

\[ \text{code} = \text{Get-Code}(CTv, v(C_i)) \]

\[ CCD = \text{CCD} \cup \text{code} \]

**end.**

In the above procedure, Get-Code(CTv, v(C_i)) is a procedure to return the binary code for attribute C, in data Di using the code table for C, and “||” denotes the concatenation of the binary strings on either side.

The categorical attribute part of a query is not always expressed in ground terms; it may contain general terms of internal nodes in the concept hierarchies. To generate query codes, the nodes of concept hierarchies are labeled with binary string(s) by propagating the leaf node labels upward and keeping the newly arrived labels in the nodes. For the categorical part of the query \( (C_j = t_j, \ldots, C_p = t_p) \), its code can be obtained by finding the categorical codes of the attributes and combining them. Generate-Cat-Query-Code(CTS, C) does this by producing a set of binary codes for the categorical part C of the query by using the code table set CTS, which contains the associations of terms and component codes.

**procedure Generate-Cat-Query-Code(CTS, C)**

**input:** code table set CTS = \{ CTv, …, CTq \}, categorical query part \( C = (C_{j_1} = t_{j_1}, \ldots, C_{p} = t_{p}) \)

**output:** a set CCD of binary codes

**begin**

**for** \( t_i \in C \)

**if** \( t_i = \text{lb}(\text{nd}) \)

\[ \text{code}_{ik} = \{ \text{Get-Code}(CT_{ji}, t_i) \} \]

**else**

\[ \text{code}_{ik} = \emptyset \]

INS = \{tn(\text{nd}) such that \( t_i = \text{lb}(\text{nd}) \) \}

**for** ln ∈ INS

\[ \text{code}_{ik} = \text{code}_{ik} \cup \text{Get-Code}(CT_{ji}, \text{lb}(\text{ln})) \]

**for** i = 1 to q

**if** \( C_i = C_{ki} \)

**end.**
CD_i = \{\text{code}_{k}\}

\text{else}
CD_i = \{\text{all possible binary codes for } C_i\}
CD = \{CD_1, CD_2, \ldots, CD_q\}
CCD = \text{Cat-Code-Combination}(CD)

end.

Cat-Code-Combination(CD) generates all possible binary codes using the allowed code segments for each categorical attribute.

procedure Cat-Code-Combination(CD)
input: a set CD = \{CD_1, CD_2, \ldots, CD_q\} of binary codes
output: a set CCD of binary codes
begin
Q = \text{Create-Queue}()
R = \text{Create-Queue}()
for cd \in CD_1
Enqueue(Q, cd)
for i = 2 to q
while (!Empty-Queue(Q))
csg = \text{Dequeue}(Q)
for cd \in CD_i
Enqueue(R, csg||cd)
while (!Empty-Queue(R))
Enqueue(Q, \text{Dequeue}(R))
CCD = \emptyset
while (!Empty-Queue(Q))
CCD = CCD \cup \text{Dequeue}(Q)
end.
The codes generated from Code-for-Categorical-Domain and Generate-Cat-Query-Code could be too long to be used as hash codes when the number of categorical attributes and their values are large. To deal with this situation, a hash function \( h(x) \) can optionally be used, which maps binary strings to shorter hash codes.

Suppose that there are three concept hierarchies and they are assigned component codes for the three categorical attributes \( C_1, C_2, \) and \( C_3, \) as shown in Figure 5. When the categorical part of a query is given as \( (C_1 = g_3, C_2 = i_2, C_3 = i_5) \), the procedure Generate-Cat-Query-Code generates the following query codes: 100110, 100111, 101010, and 101011.

3.3 Locality-Sensitive Hashing for Numerical Attributes Using Fuzzy Ball Queries
A fuzzy ball indicates a cluster on a numerical data space, centered at a point, for which membership to the cluster decreases as distance from the center increases. To locate data points within a fuzzy ball, a locality-sensitive hashing technique is used, employing the approach proposed by Lee\(^{22}\) in which the extended boundary LSH allows the fuzzy balls centered near a space partitioning boundary to retrieve the data points within their influence by enforcing extension bounds.

Code-for-Numerical-Domain(\( H, D, \delta \)) determines the hash code for a data object \( D \) using the hyperplane vectors \( H \) with extension bound \( \delta \). The hyperplane vectors are randomly selected from \((p+1)\)-dimensional space, where \( p \) is the dimension of the numerical space in which the numerical parts of the data lie, and \( \delta \) is the parameter controlling the extension bound of the space-partitioning hyperplanes.

procedure Code-for-Numerical-Domain(\( H, D, \delta \))
input: hyperplane vectors \( H = \{h_1, h_2, \ldots, h_n\} \), a data object \( D \), extension bound \( \delta \)
output: a set Code of codes
begin
Code = \{\emptyset\}
for \( i = 1 \) to \( n \)
tmpCode = \{\emptyset\}
if ds(h_i, NV(D)) > \delta
\text{for } cd \in \text{Code}
end.

Figure 5. Concept hierarchies and code segments for three categorical attributes C1, C2, and C3.
tmpCode = tmpCode \cup \{ cd \}\}$

\textbf{if} \ ds(h, NV(D)) < -\delta \\
\textbf{for} \ cd \in \text{Code} \\
\quad \text{tmpCode} = \text{tmpCode} \cup \{ cd \}\$ \\
\textbf{if} \ -\delta \leq ds(h, NV(D)) \leq \delta \\
\textbf{for} \ cd \in \text{Code} \\
\quad \text{tmpCode} = \text{tmpCode} \cup \{ cd \}\$ \\
\text{Code} = \text{tmpCode}$

\textbf{end.}$

In the above procedure, $ds(h, NV(D))$ is a function to compute the distance from point $NV(D)$, corresponding to the numerical part of data object $D$, to hyperplane $h$. It results in either positive or negative distance depending on which side of the hyperplane the data object is located.

Figure 6. Code assignment in the numerical domain

Figure 6 shows a code assignment for three data points $D_1$, $D_2$, and $D_3$ with respect to two partitioning hyperplanes $h_1$ and $h_2$. A hyperplane partitions the space into two separate subspaces, and assigns 1 to one side and 0 to the other. The hyperplanes have extension bounds that cover the narrow band of width $2\delta$. A data point within an extension bound is treated as being on both sides of its hyperplane. The two hyperplanes map the data points into binary codes of length 2 as follows: $D_1$ is mapped into $10$, $D_2$ into $10$ and $11$, and $D_3$ into $11$.

\textbf{Hash-Data-for-Numerical-Domain}(DS, H, \delta)$

distributes the data set $DS$ into $p$ buckets based on the numerical parts of the data.

\textbf{procedure} \text{Hash-Data-for-Numerical-Domain}(DS, H, \delta) \\
\textbf{input:} data set $DS$, hyperplane vector $H = \{ h_1, h_2, \ldots, h_n \}$, maximum distance threshold $\delta$ \\
\textbf{output:} set of buckets $B = \{ B_1, B_2, \ldots, B_p \}$ where $p = 2^n$

\textbf{begin} \\
\quad \textbf{for} \ B \in B \\
\quad \quad B_i = \emptyset \\
\textbf{for} \ D \in DS \\
\quad \text{Code} = \text{Code-for-Numerical-Domain}(H, D, \delta) \\
\textbf{end.}$

3.4 Categorical Bucket Selection for Queries

When a query $Q$ is given, its categorical part $CV(Q)$ is extracted. Then, its binary map is determined using $\text{Code-for-Categorical-Domain}(CTS, CV(Q))$. When the system uses a rehash function $h$, the hash function is applied to the codes thus obtained, to get the identifiers of the buckets.

3.5 Numerical Bucket Selection for Queries

A fuzzy ball for the numerical part is described with a center for the ball and a membership function. The LHS for the numerical domain is constructed with a set $H$ of hyperplanes. From the membership function and the minimum degree of satisfaction specified in the query, we can determine the maximum allowed distance $d_m$ from the center of the ball to the data. In order to determine the buckets to be examined, we compute the distances $ds(c, h)$ of the center $c$ to each hyperplane $h$. When the distance is farther than the maximum allowed distance $d_m$ plus the extension bound $\delta$, i.e., $ds(c, h) > d_m + \delta$, the neighboring bucket to the hyperplane should be simultaneously examined.

\textbf{Get-Num-Bucket}(H, F, \delta, \theta)$

determines the buckets containing data objects that might be covered by a fuzzy ball $FB$ with the minimum satisfaction degree $\theta$.

\textbf{procedure} \text{Get-Num-Bucket}(H, F, \delta, \theta) \\
\textbf{input:} hyperplane vector $H = \{ h_1, h_2, \ldots, h_n \}$, fuzzy ball $FB(ct, LT)$, extension bound $\delta$, data object $D$, minimum satisfaction degree $\theta$ \\
\textbf{output:} a set $\text{Code}$ of codes 

\textbf{begin} \\
\quad d_m = \text{max-allowed-distance}(LT, \theta) \\
\quad \text{Code} = \text{Query-Code-for-cDomain}(NV(D), H) \\
\quad Q = \text{Create-Queue}() \\
\quad R = \text{Create-Queue}() \\
\quad \text{Enqueue}(Q, \text{Code}) \\
\quad \textbf{for} \ i = 1 \text{ to } n \\
\quad \quad \text{if} \ ds(h, ct) > d_m + \delta \\
\quad \quad \quad \textbf{while} \ \text{(!Empty-Queue}(Q)) \\
\quad \quad \quad \quad \text{cs1} = \text{Dequeue}(Q) \\
\quad \quad \quad \quad \text{cs2} = \text{Toggle-position}(cs1, i)$
In the above procedure, \( \text{max-allowed-distance}(LT, \theta) \) computes the distance at which the membership degree for \( LT \) is \( \theta \). \( \text{Toggles-position}(cs1, i) \) copies \( cs1 \) and flips its \( i \)-th position from 1 to 0 or vice versa.

\( \text{Query-Code-for-cDomain}(NV(D), H) \) computes the binary code for the bucket(s) to which the numerical attribute part of data object \( D \) is mapped.

\[\text{Enqueue}(R, cs1)\]
\[\text{while} (!\text{Empty-Queue}(R)) \]
\[\text{Enqueue}(Q, \text{Dequeue}(R))\]
\end{verbatim}

In the above procedure, \( side(h, NV(D)) \) returns 1 if \( NV(D) \) is on the side designated “1” of hyperplane \( h \); otherwise, 0.

### 3.6 Candidate Data Set Selection

When a query is initiated, the candidate data sets to be examined are efficiently determined by selecting proper LSH buckets. The numerical part of the query is used to determine the target bucket(s) \( NB \) of the numerical LSH; the categorical part is used to determine the target bucket(s) \( CB \) of the categorical LSH. The data set belonging to their intersection \( NB \cap CB \) becomes the candidate data set for thorough examination. For an efficient intersection operation, the Bloom filter can be used, which never fails to miss the true candidate.

### 3.7 Similarity Examination between Query and Candidates

The similarity is computed for the numerical and categorical parts separately. For the numerical part, the distance between the data point and the center of the fuzzy ball is computed, and its membership degree to the fuzzy ball is then determined. When the membership degree is less than the threshold specified in the query, the data point is ignored. For a data point that meets the constraints for the numerical part, the satisfaction degree for the categorical part of the query is evaluated. When a fuzzy concept hierarchy is given, the degree of satisfaction for a categorical value to a general term is set to be the minimum of the weights on the path from the general term to the categorical value.

### 4. Experimental Results

To show the effectiveness of the proposed method, we performed some experiments using synthetic data. The synthetic data set had 10 numerical attributes and 10 categorical attributes. Each attribute on the numerical domain had range \([0,10] \). For the categorical attributes, the numbers of ground terms ranged from 10 to 15, and the concept hierarchies on the ground terms were constructed to have a height of 3 or 4, in which there was a single top node and each node had fewer neighboring upper levels than nodes at the level below it. The internal nodes of the concept hierarchies were labeled with unique names. A data set of size 20,000,000 was generated, in which each data object consisted of a randomly sampled point from \([0,10]^n \) and 10 categorical values, each of which was sampled from the ground categorical attribute values of a categorical attribute. For the experiments, 100 sample queries were generated as follows: For the numerical part of the queries, a fuzzy ball was specified by randomly sampling a point from \([0,10]^n \) as its center, a value from \([0.1,1] \) as the standard deviation \( s \) of the Gaussian membership function of Eq. (9), and a value from \([0.5,1] \) as the threshold for degree of satisfaction.

\[
\mu_{\text{gauss}}(d,s)(x) = \exp(-(a^2/s^2))
\]  

(9)

Two locality-sensitive hashing settings were implemented: The first setting (Set I) used 10 hyperplanes to realize a hyperplane-based LSH for the numerical space and a 10-bit categorical LSH, and the second setting (Set II) used 15 hyperplanes for the numerical space LHS and a 15-bit categorical LSH. The hyperplanes for LSH were also randomly generated, and the extension bound was set to 0.3. The rehash functions of the categorical LSH were defined as a mapping function that performs a permutation of the bit string and then applies the modulo to the output bit size.

In the experiments, we evaluated the performance characteristics using the following measures: the portions of the data set to be examined as the candidate when only the numerical LSH was used, when only the categorical
LSH was used, and when both LSHs were used (Tables 1 and 2); and the average number of numerical buckets to be examined in each experimental setting (Table 3).

**Table 1.** The portion of the data set to be examined for similarity comparison at the final stage when only one of two LSH types was used

|                  | Only Numerical LSH | Only Categorical LSH |
|------------------|--------------------|----------------------|
| Set I            | 0.09124%           | 2.04310%             |
| Set II           | 0.00348%           | 0.25621%             |

**Table 2.** The portion of the data set to be examined for similarity comparison at the final stage when both numerical and categorical LSH types were used

| Portion of Data to Be Examined |                  |
|--------------------------------|------------------|
| Set I                          | 0.02712%         |
| Set II                         | 0.00084%         |

**Table 3.** The average number of numerical buckets examined

| Number of Buckets Examined |                  |
|----------------------------|------------------|
| Set I                      | 1.42             |
| Set II                     | 2.53             |

Table 1 implies that as the more the number of buckets increase, the smaller portion of the data set is examined at the expense of storage overhead for bucket management. The results shown in Table 2 indicate that the dual LSH method is more efficient than either numerical LSH or categorical LSH method in the perspective to the number of examined data at the final stage. Table 3 shows that additional numerical buckets are examined depending on the threshold of the degree of satisfaction, and as the hash code length increases, the bucket size decreases and hence more additional buckets are examined.

**5. Conclusions**

Similarity search and nearest neighbor search are well-known fundamental operations in various data processing tasks. Large volumes of high-dimensional data spaces having both numerical and categorical attributes are now available in many application domains. To support approximate but fast search for such data, we have proposed a new LSH technique based on the dual LSH approach. We believe this work presents the following contributions: First, it introduces a flexible query scheme for data having both numerical and categorical attributes by defining the notion of fuzzy balls and incorporating fuzzy concept hierarchies in categorical attributes to support the queries with general categorical terms. Second, a new categorical LSH technique is introduced that incorporates concept hierarchies and generates reduced-length hash codes by applying an additional rehash. Third, the proposed method optionally employs the Bloom filter to reduce the time to compute the intersection of buckets from both numerical and categorical LSHs for large volumes of data.

From the experimental studies, we observe that the proposed method shows the desired and expected behaviors: The proposed dual LSH outperforms both the numerical LSH and the categorical LSH in the perspective of the number of examined data objects for finding similar data. This study considers only the fuzzy balls for the entire numerical space, and there yet remains to develop an LSH method to handle fuzzy balls defined on a numerical subspace.

**6. Acknowledgment**

This research was supported by Basic Science Research Program through the National Research Foundation (NRF) of Korea (grant no.: 2015R1D1A1A01061062).

**7. References**

1. Zezula P, Amato G, Dohnal D, Batko M. Similarity search: The metric space approach. Advances in Database Systems. Purdue University, 2006.
2. Omohundro SM. Five balltree construction algorithms. Technical Report. International Computer Science Institute; 1989.
3. Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM. 1975; 18(9):509-17.
4. Ruan Da. Fuzzy Set Theory and Advanced Mathematical Applications. Springer Science & Business Media; 2013.
5. Cormen TH, Leiserson CE, Rivest RL. Introduction to Algorithms. The MIT Press; 2001.
6. Sugguna S, Ranjith KC, Sheela JC. State of the Art: A Summary of Semantic Image and Video Retrieval Techniques. Indian Journal of Science and Technology. 2015; 8(35):1-12.
7. Lee J, Kim DW. Ranking Tag Pairs for Music Recommendation Using Acoustic Similarity. International Journal of Fuzzy Logic and Intelligent Systems. 2015; 15(3):159-165.
8. Ganeshkumar K, Arivazhagan D. Generating a Digital Signature Based on New Cryptographic Scheme for User Authentication and Security. Indian Journal of Science and Technology. 2014; 7(6):1-5.
9. Indyk P, Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing; 1998. p. 604-13.
10. Kim YG, Lee KM. Association-based Outlier Detection for Mixed Data. Indian Journal of Science and Technology. 2015; 8(25):1-6.
11. Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. Proceedings of the 25th VLDB Conference; 1999, p. 518-29.
12. Lee KM. Locality-sensitive hashing techniques for nearest neighbor search. International Journal of Fuzzy Logic and Intelligent Systems. 2012; 12(4):308-318.
13. Rajaraman A, Ullman JD. Mining of Massive Datasets. Cambridge University Press. 2011.
14. Nister D, Stewenius H. Scalable recognition with a vocabulary tree. Proceedings of IEEE CVPR; 2006. p. 2161-68.
15. Shahkarovich G, Viola P, Darrell T. Fast pose estimation with parameter sensitive hashing. Proceedings of IEEE Int. Conference on Computer Vision; 2003. p. 750-57.
16. Salakhutdinov RR, Hinton GE. Semantic hashing. International Journal of Approximate Reasoning. 2009; 50(7):969-78.
17. Jin Z, Li C, Lin Y, Cai D. Density sensitive hashing. IEEE Transactions on Cybernetics. 2014; 44(8):1362-71.
18. Weiss Y, Torralba A, Fergus R. Spectral hashing. Proceedings of Neural Information Processing Systems; 2008. p. 1753-60.
19. Lee KM. Locality-sensitive hashing for data with categorical and numerical attributes using dual hashing. International Journal of Fuzzy Logic and Intelligent Systems. 2014; 14(2):308-18.
20. Bishop CM. Pattern Recognition and Machine Learning. Springer, 2006.
21. Scholkopf B, Smola AJ. Learning with kernels. The MIT Press. 2002.
22. Lee KM. Locality sensitive hashing with extended partitioning boundaries. Applied Mechanics and Materials. 2013; 321-324:804-07.
23. Mendel JM. On a novel way of processing data that uses fuzzy sets for later use in rule-based regression and pattern classification. International Journal of Fuzzy Logic and Intelligent System. 2014; 14(1):1-7.
24. Lee KM, Lee KM, Lee CH. Statistical cluster validity indexes to consider cohesion and separation. Proceedings of 2012 International Conference on Fuzzy Theory and its Applications; 2012. p. 228-32.
25. Lee KM. Mining generalized fuzzy quantitative association rules with fuzzy generalization hierarchies. Proceedings of IFSA World Congress and 20th NAFIPS International Conference; 2001. p. 2977-82.
26. Lee KM, Lee KM. A Locality Sensitive Hashing Technique for Categorical Data. Applied Mechanics and Materials. 2013; 241-244:3159-64.