Issue 7 volume 6 Nov.- Dec. 2017
ISSN 2249-9954

NEAREST KEYWORD SET SEARCH ALGORITHM FOR TEXT-RICH MULTI DIMENSIONAL DATA SETS

ASMA BEGUM [#1] **, Prof. B.PRAJNA**[#2]

[#1] M.Tech Scholar , Department of Computer Science and System Engineering,
Andhra University College of Engineering (A), Visakhapatnam, AP, India.

[#2] Professor, Department of Computer Science and System Engineering,
Andhra University College of Engineering (A), Visakhapatnam, AP, India.

ABSTRACT

Now a day's data mining has become one of the most fascinating domains in each and every field like medical, shopping, business, MNC companies, information technology and a lot more. As we all know that the main goal of data mining is to extract the valuable information from large data sets, in order to retrieve the desired result as an output. In this paper we mainly try to extract the large numbers of useful keywords from a document which has some meaningful information about that specific topic. Here we will choose a conversation file as input which contains a set of useful keywords related to that conversation topic. Initially in this paper we will try to extract one or more keywords which are almost useful for extracting the whole keywords that are available in the document. If we take a small piece of document also it contains a set of words, which are potentially related to the several topics among that conversation document. Most of existing works are mainly concentrated on topic modeling and the evolution of individual topics, while most of the sequential relations of topics in successive documents published by a specific user are ignored. In this paper, we mainly try to design a multi-dimensional datasets in which a set of keywords is represented as a data point. In this paper, we mainly try to design and analyze the nearest keyword set (termed as NKS) queries on very text data sets having multi-dimensional features. Here for an NKS query, the user need to provide a set of keywords, and the result of the user query may constitute k sets of data points each of which contains all the query keywords and forms one of the top-k tightest clusters in the multi-dimensional space. By conducting various experiments on our proposed NKS technique, our simulation results clearly tell that our proposed method is very best in finding the nearest keyword set search in a very less time than compared with several primitive methods.

Key Words:
Keyword Search, Multi-Dimensional Datasets, Keyword Set, Data Point, Clustering Space, Query Evolution.

I. INTRODUCTION

Data mining is the process of extracting useful or structured information from a raw or un-structured data. Generally this is used mainly in performing operations like insurance sector, bank and retail sector, hospital for identifying diseases, shopping malls to calculate the priority of items that were sold [1]. Data mining processes have required an integration of techniques from multiple disciplines such as statistics, machine learning, database technology, pattern recognition, neural networks, information retrieval and spatial data analysis. The process of data mining involves a keen observation with a set of algorithms to accomplish different tasks. All these algorithms attempt to fit a model to the given data set. All the data mining algorithms are mainly used to examine the input data and then determine a model that is very closest to the characteristics of the data being examined [2].

Generally the data mining algorithms can be classified into different ways based on the following three things like:

1. **Model Based:** The purpose of the algorithm is to fit a model to the required data.

2. **Preference Based:** This is based on the preferences or criteria that are being used.

3. **Search Based:** In this method, we mainly find out the search techniques that are being used.

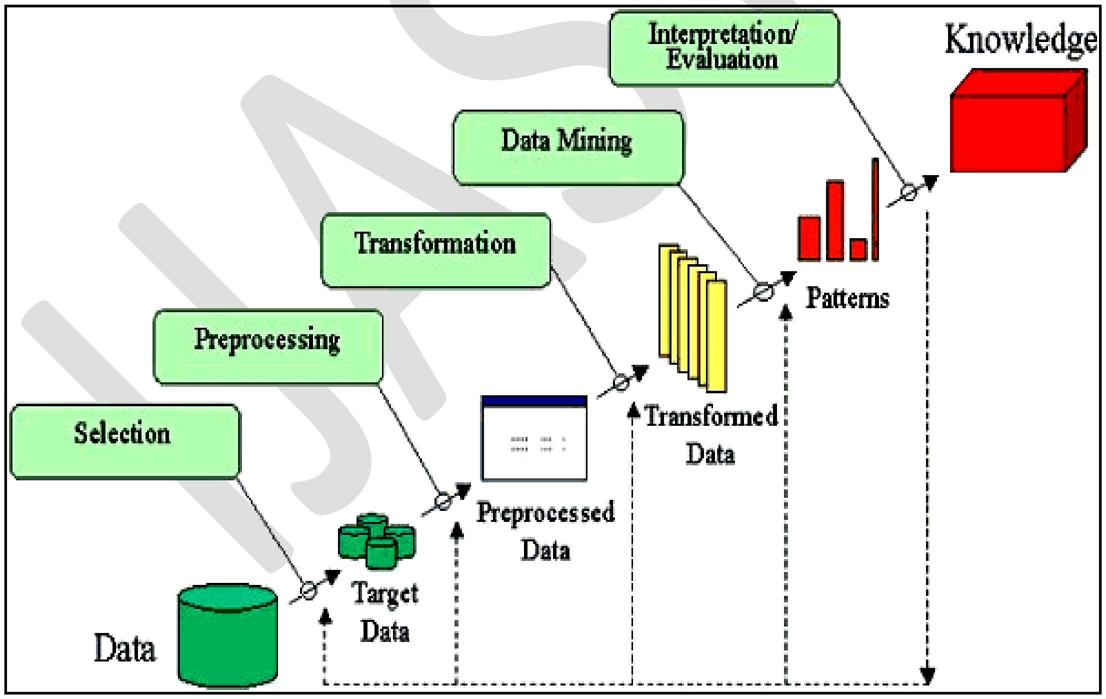

Figure 1.Represents the Basic Architecture of a Data Mining As a Core Method In The Process of Knowledge Discovery

From the above figure 1, we can clearly find out that for the process of data mining the data which should be taken as input will be collected from various resources like world wide web, Database Schema, Data Warehouse and other data repositories. Once the input data is collected it will be then given to the process of data cleaning. Here in this data cleaning process the data will be cleaned and it will be processed in order to identify if there are any un-supervised data available in that input data [3]. Once that data cleaning is completed now it will be processed for further iterations for data mining engine in order to process the knowledge base .Once the process of pattern evaluation is done then the data will be in turn converted into GUI.This graphical user interface is one which will give us the visualization of output [4].

Generally the objects like images, documents or products that may define the chemical compounds and so on are mostly defined and visualized by a collection of relevant features, and are commonly represented as data points in a multi-dimensional feature space. For example, if we take images into consideration, initially images are represented using the color attribute and the features are extracted as color feature vectors which will try to explain the description about that image. For a better performance in this proposed thesis, we try to consider multi-dimensional datasets which contain a set of keywords to be extracted for search. Here the query keywords are also known as data points that clearly define the functionality of the item and we also try to analyze and design a nearest keyword set (referred to as NKS) queries on text-rich multi-dimensional datasets. An NKS query is a set of user-provided keywords, and the result of the query may include k sets of data points each of which contains all the query keywords and forms one of the top-k tightest clusters in the multi-dimensional space. Figure. 2 define an NKS query over a set of two-dimensional data points. Each point is tagged with a set of keywords. For a query Q = {a, b, c}, the set of points {7, 8, 9} contains all the query keywords {a, b, c}, and forms the tightest cluster compared with any other set of points covering all the query keywords. Therefore, the set {7, 8, 9} is the top-1 result for the query Q.

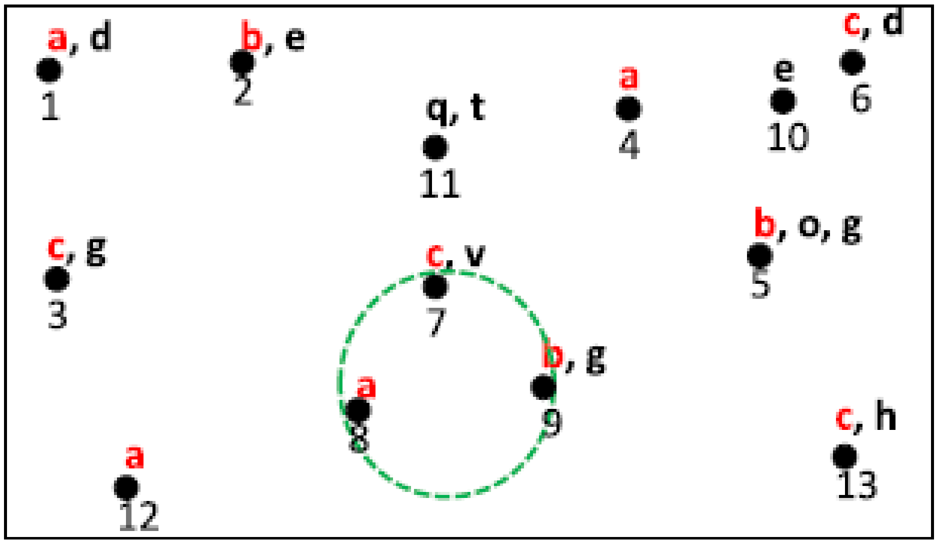

Figure 2.Represents The Example Of An Nks Query On A Keyword Tagged Multi-Dimensional Dataset. The Top-1 Result for Query Fa; B; Cg Is the Set of Points {7; 8; 9}

II. BACKGROUND KNOWLEDGE

In this section we mainly discuss about the background work that was carried out in finding the work that is related to Index structure and flow of execution of ProMiSH algorithm. In this section we mainly discuss about the basic data mining algorithms that are classified and various data mining models. Now let us discuss about this in detail:

ABOUT TEXT MINING

The term **text mining**, sometimes also referred to as text data mining, roughly equivalent to text analytics, refers to the process of deriving high-quality information from text data set. Normally from statistical pattern learning we mainly try to derive the patterns and trends that are calculated from the high quality of information. As we all know that the process of extracting or structuring the input text by identifying the main features and removing the ir-related data from that main document and finally convert the document in a structured way is known as text mining[5]. In this paper we use a word like high quality information, which is typically derived through a set of patterns and trends that are used in pattern learning applications. Also the term high quality in this paper clearly states that combination of some relevance and interestingness for the topic that was available in that conversation file. In the primitive text mining ,there are a possibilities like to scan a set of documents written in a natural language and either model the document set for predictive classification purposes or populate a database or search index with the information extracted [6],[7].

DATA PREPROCESSING METHODS

Generally this stage is very important in our current application as the data preprocessing plays a very vital role in the current application. In this section we mainly discuss about the methods that are used in the current applications. The various types of applications are explained in detail by the following figure 2.

As we all know that if we take any input for the data mining applications those are treated as a Raw data and which is almost have noise, missing values, and inconsistency of data. The quality of the data always affects the data mining results.As in our project we will take documents having various conversations of a topic,where each and every document contains various keywords of same topic[8].Here some keywords belongs to one topic and some don't belong to that exact topic but included in that conversation file. So in order to improve the quality of the data and, consequently, of the mining results the input data or raw data should be pre-processed so as to improve the efficiency and ease of the mining process [9].

Data preprocessing is considered as the one of the most critical steps in a data mining process which mainly deals with the preparation of initial data set followed by transformation of the initial dataset. Data preprocessing methods are divided into following four categories as described below:

A. Data Cleaning method
B. Data Integration method
C. Data Transformation method
D. Data Reduction method

Of all the above four categories each and every category has its individual advantages and usage.Initailly the raw data what we take as input is entered into data cleaning method and once that category processes the data,the data will be cleaned and displayed with no errors. Now the data which is processed is now entered for transformation method, where a set of data is integrated based on any common attribute. Once the data is transformed now it will be entered into the data reduction category which is the final category. In this category the data will be represented in the form of tabular way with all the corresponding data into the table cells.

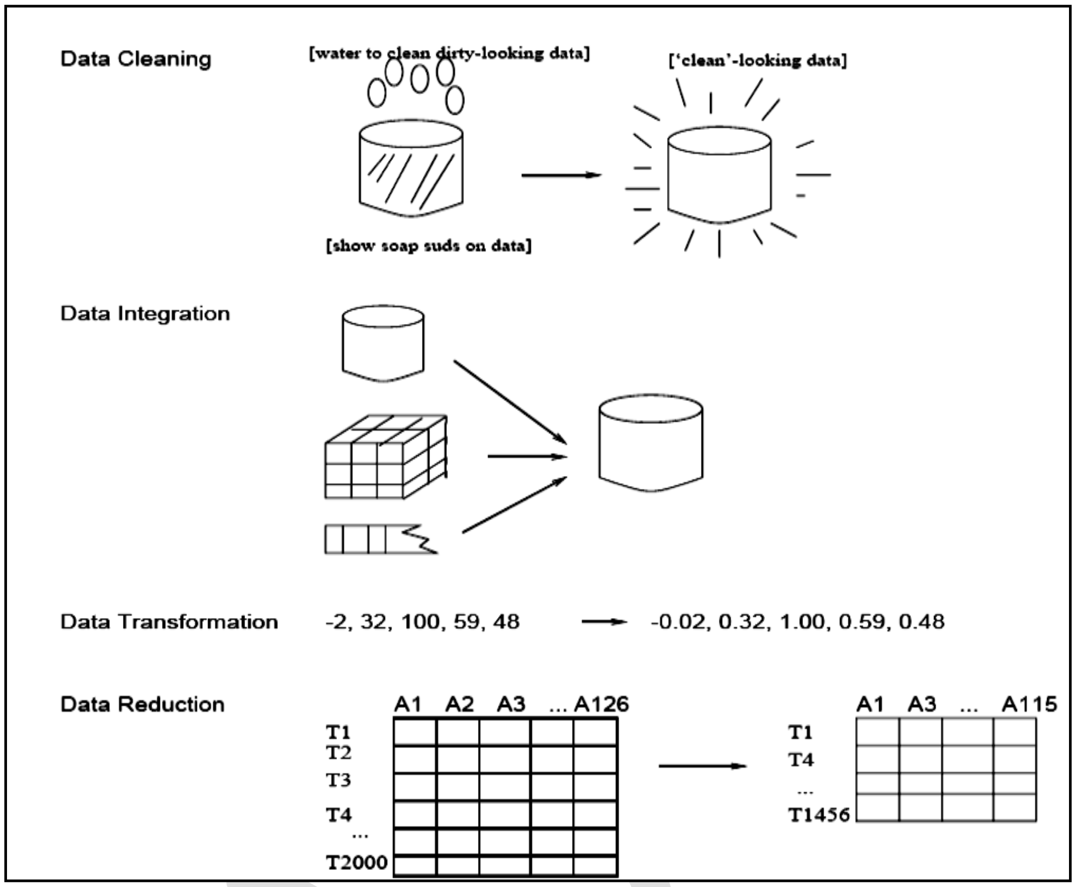

Figure 3.Represents the Various Forms of Data Pre-Processing

In the aspect of our proposed thesis for finding Index structure and flow of execution of ProMiSH algorithm, this is mainly used for identifying a set of keywords among the input text documents and try to add those keywords or data points into the database[10]. So that once if the data user want to search the data ,any of the multiple data points can be given as a input keyword, so that the data will be searched based on that data point .

III. THE PROPOSED PROJECTION AND MULTI-SCALE HASHING (ProMiSH) TO ENABLE THE FAST PROCESSING FOR THE NKS QUERY

In this section we mainly discuss about the proposed ProMiSH mining for mining the data based on content, topic, keyword and date based approach. Now let us discuss about this proposed ProMiSH method in detail as follows:

PRELIMINARY KNOWLEDGE

The main processing framework for the proposed task of finding nearest keyword set search in multi dimensional data set is clearly shown in below Figure. 3. This proposed algorithm is mainly divided into two main components:

1.INVERTED INDEX IKP:

The first component is an inverted index referred to as Ikp. In Ikp, we treat keywords as keys, and each keyword points to a set of data points that are associated with the keyword. We build Ikp as follows. (1) For each ,we create a key entry in Ikp, and this key entry points to a set of data points (2) We repeat (1) until all the keywords are processed.

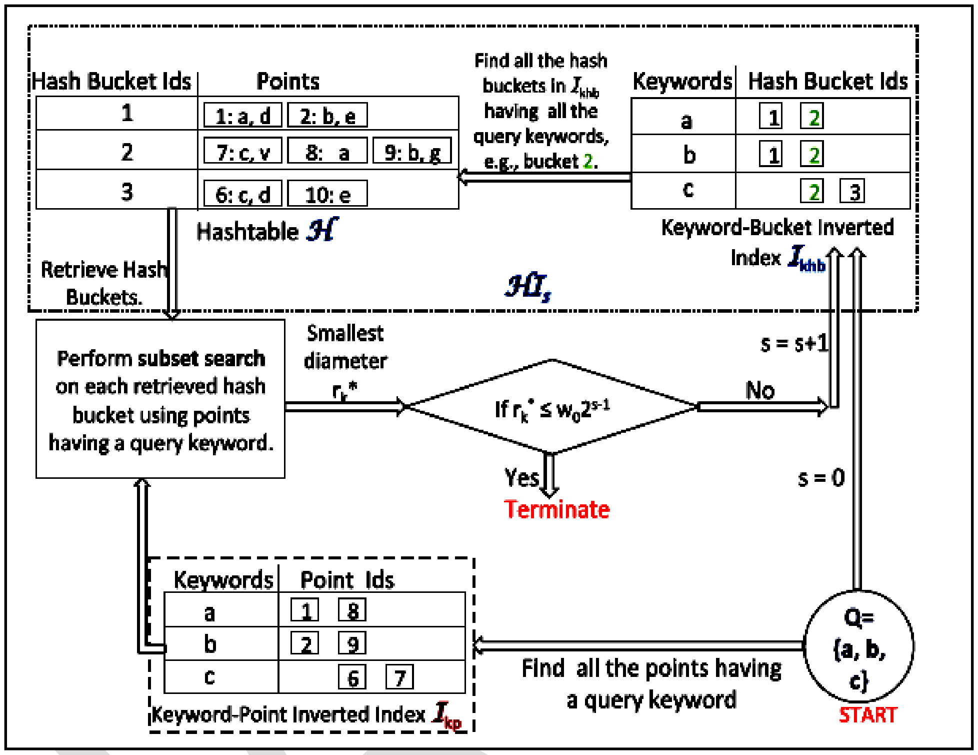
Hash Bucket Ids
Points
1
1: a, d
2: b, e
2
7: c, v
8: a
9: b, g
3
6: c, d
10: e
Hashtable $\mathcal{H}$
Find all the hash buckets in I_{khb} having all the query keywords, e.g., bucket 2.
Keywords
Hash Bucket Ids
a
1
2
b
1
2
c
2
3
Keyword-Bucket Inverted Index I_{khb}
Retrieve Hash Buckets.
$\mathcal{HI}_s$
Perform subset search on each retrieved hash bucket using points having a query keyword.
Smallest diameter r_k^*
If $r_k^* \le w_0 2^{s-1}$
s = s+1
No
Yes
Terminate
s = 0
Keywords
Point Ids
a
1
8
b
2
9
c
6
7
Keyword-Point Inverted Index I_{kp}
Find all the points having a query keyword
Q= {a, b, c}
START

Figure 4.Represents the Detailed Processing of Index Structure and Flow of Execution of ProMiSH Algorithm

2.HASH TABLE-INVERTED INDEX PAIRS HI:

The second component consists of multiple hash tables and inverted indexes referred to as HI. HI is controlled by three parameters:

(1) Index level,
(2) Number of random unit vectors, and
(3) Hash table size.

All the three parameters are non-negative integers. These three parameters control the construction of HI.

THE EXACT SEARCH ALGORITHM

In exact search we are using ProMiSH-E algorithm that finds exact search results for NKS queries. In this algorithm hash indexes are used to store keys related to keywords [11]-[13]. When user searching for file by entering search key it will compare that keyword with hash table keys and exact file will be retrieved by using following algorithms.

ALGORITHM 1.ProMiSH-E

```
Q:Query keywords; k:number of top results; $w_{0:}$initial bin-width
1: P Q ← [e([ ], +∞)]: priority queue of top-k results
2: HC: hashtable to check duplicate candidates
3: BS : bitset to track points having a query keyword
4: for all o ∈ ∪∀$V_Q$∈Q$I_{kp}[V_Q]$ do
5: BS[o] ← true /* Find points having query keywords*/
6: end for
7: for all s ∈ {0,…., L − 1} do
8: Get HI at s
9: E[ ] ← 0 /* List of hash buckets */
10: for all vQ ∈ Q do
11: for all bId ∈ $I_{khb}[V_Q]$ do
12: E[bId] ← E[bId] + 1
13: end for
14: end for
15: for all i ∈ (0, ..., SizeOf(E)) do
16: if E[i] = SizeOf(Q) then
17: F 0 ← ∅ /* Obtain a subset of points */
18: for all o ∈ H[i] do
19: if BS[o] = true then
20: F’ ← F’∪ o
21: end if
22: end for
23: if checkDuplicateCand(F’, HC) = false then
24: searchInSubset(F’, PQ)
25: end if
26: end if
27: end for
28: /* Check termination condition */
29: if PQ[k].r ≤ w02 s−1 then
30: Return PQ
31: end if
32: end for
33: /* Perform search on D if algorithm has not terminated */
34: for all o ∈ D do
35: if BS[o] = true then
36: F’← F’∪ o
37: end if
38: end for
39: searchInSubset(F’ , PQ)
40: Return PQ
```

Algorithm 1 details the steps of ProMiSH-E.
Step 1: Initialize priority queue of top-k results
Step 2: It maintains hash table (HC) to check duplicate candidates
Step 3: It maintains a bit set BS
Step 4: For each vQ ∈ Q, ProMiSH-E retrieves the list of points corresponding to vQ from Ikp.
step 5: In each point o in the retrieved list, ProMiSH-E marks the bit corresponding to o's identifier in BS as true.
Step 6: end for
Step 7-9: Finds all the points in D which are tagged with at least one query keyword. Next, the search continues in the HI structures, beginning at s=0. For any given scale s, ProMiSH-E accesses the HI structure created at the scale.
Step 10-11: it retrieves all the lists of hash bucket ids corresponding to keywords in Q from the inverted index Ikhb.
Step 12-16: An intersection of these lists yields a set of hash buckets each of which contains all the query keywords.
Step 17-22: For each selected hash bucket, it retrieves all the points in the bucket from the hash table H. It filters these points using bitset BS to get a subset of points F'.
Step 23: Subset F' contains only those points which are tagged with at least one query keyword and is explored further. Subset F' is checked whether it has been explored earlier or not using checkDuplicateCand.
Step 24: It performs a search on it using searchInSubset.
Step 25: End if
Step 26: End if
Step 27: End for
Step 28: Checking termination condition
Step 29-31: If it does not terminate after exploring the HI structure at the scales, then the search proceeds to HI at the scale (s + 1). It terminates when the kth smallest diameter rk in P Q becomes less than or equal to half of the current binwidth $w=w_0 2^{s-1}$.
Step 32: End for
Step 33: Perform search on D(dataset) if algorithm has not terminated.
Step 34-39: If it fails to terminate after exploring HI at all the scale levels s ∈ {0, ..., L − 1}, then it performs a search in the complete dataset D.

ALGORITHM 2.CheckDuplicateCand

The algorithm CheckDuplicateCand is using hash table HC to check duplicates for subsets.

```
In: F' : a subset; HC: hash table of subsets
1: F' ← sort(F' )
2: pr1: list of prime numbers; pr2: list of prime numbers;
3: for all o ∈ F' do
4: pr1 ← randomSelect(pr1); pr2 ← randomSelect(pr2)
5: h1 ← h1 + (o × pr1); h2 ← h2 + (o × pr2)
6: end for
7: h ← h1h2;
8: if isEmpty(HC[h])=false then
```

```
9: if elementWiseMatch(F' , HC[h]) = true then
10: Return true;
11: end if
12: end if
13: HC[h].add(F' );
14: Return false;
```

Algorithm checkDuplicateCand (Algorithm 2) uses a hashtable HC to check duplicates for a subset F 0 . Points in F 0 are sorted by their identifiers. Two separate standard hash functions are applied to the identifiers of the points in the sorted order to generate two hash values in steps (2-6). Both the hash values are concatenated to get a hash key h for the subset F 0 in step 7. The use of multiple hash functions helps to reduce hash collisions. If HC already has a list of subsets at h, then an element-wise match of F 0 is performed with each subset in the list in steps (8-9). Otherwise, F 0 is stored in HC using key h in step 13.

ALGORITHM 3.SearchInSubset

Algorithm SearchInSubset is shows how the groups are ordered

```
In: F': subset of points; Q: query keywords; q: query size
In: P Q: priority queue of top-k results
1: $r_k \leftarrow$ P Q[k].r /* kth smallest diameter */
2: SL $\leftarrow$ [(V, [ ])]: list of lists to store groups per query keyword
3: for all V ∈ Q do
4: SL[V] $\leftarrow$ {∀o ∈ F' : o is tagged with V} /* form groups */
5: end for
6: /* Pair wise inner joins of the groups*/
7: AL: adjacency list to store distances between points
8: M $\leftarrow$ 0: adjacency list to store count of pairs between groups
9: for all ($V_i$, $V_j$ ) ∈ Q such that $i \leq q, j \leq q, i < j$ do
10: for all o ∈ SL[$V_i$] do
11: for all o' ∈ SL[$V_j$ ] do
12: if ||o – o' ||2 $\leq r_k$ then
13: AL[o, o' ] $\leftarrow$ $||o – o'||_2$
14: M[$V_i$, $V_j$ ] $\leftarrow$ M[$V_i$, $V_j$ ] + 1
15: end if
16: end for
17: end for
18: end for
19: /* Order groups by a greedy approach */
20: curOrder $\leftarrow$ [ ]
21: while Q != Ø do
22: ($V_i$, $V_j$ ) $\leftarrow$ removeSmallestEdge(M)
23: if $V_i$ !∈ curOrder then
24: curOrder.append($V_i$); Q $\leftarrow$ Q \ $V_i$
25: end if
26: if $V_j$ !∈ curOrder then
27: curOrder.append($V_j$ ); Q $\leftarrow$ Q \ $V_j$
```

```
28: end if
29: end while
30: sort(SL, curOrder) /* order groups */
31: findCandidates(q, AL, P Q, Idx, SL, curSet, curSetr, rk)
```

Algorithm 3 shows how the groups are ordered. The kth smallest diameter rk is retrieved form the priority queue P Q in step 1. For a given subset F 0 and a query Q, all the points are grouped using query keywords in steps (2-5). A pairwise inner join of the groups is performed in steps (6-18). An adjacency list AL stores the distance between points which satisfy the distance threshold rk. An adjacency list M stores the count of point pairs obtained for each pair of groups by the inner join. A greedy algorithm finds the order of the groups in steps (19-30). It repeatedly removes an edge with the smallest weight from M till all the groups are included in the order set corroder. Finally, groups are sorted using curOrder in step 30.

ALGORITHM 4.FindCandidates

```
In: q: query size; SL: list of groups
In: AL: adjacency list of distances between points
In: P Q: priority queue of top-k results
In: Idx: group index in SL
In: curSet: an intermediate tuple
In: curSetr: an intermediate tuple's diameter
1: if Idx ≤ q then
2: for all o ∈ SL[Idx] do
3: if AL[curSet[Idx-1], o] ≤ rk then
4: newCurSetr ← curSetr
5: for all o 0 ∈ curSet do
6: dist ← AL[o, o 0 ]
7: if dist ≤ rk then
8: f lag ← true
9: if newCurSetr < dist then
10: newCurSetr ← dist
11: end if
12: else
13: f lag ← false; break;
14: end if
15: end for
16: if flag = true then
17: newCurSet ← curSet.append(o)
18: rk ← findCandidates(q, AL, P Q, Idx+1, SL, newCurSet, newCurSetr, rk)
19: else
20: Continue;
21: end if
22: end if
23: end for
```

```
24: return rk
25: else
26: if checkDuplicateAnswers(curSet, P Q) = true then
27: return rk
28: else
29: if curSetr < P Q[k].r then
30: P Q.Insert([curSet, curSetr])
31: return P Q[k].r
32: end if
33: end if
34: end if
```

We find results by nested loops as shown in Algorithm 4 (findCandidates). Nested loops are performed recursively. An intermediate tuple curSet is checked against each point of group SL[Idx] in steps (2-23). First, it is determined using AL whether the distance between the last point in curSet and a point o in SL[Idx] is at most rk in step 3. Then, the point o is checked against each point in curSet for the distance predicate in steps (5-15). The diameter of curSet is updated in steps (9-11). If a point o satisfies the distance predicate with each point of curSet, then a new tuple newCurSet is formed in step 17 by appending o to curSet. Next, a recursive call is made to findCandidates on the next group SL[Idx+ 1] with newCurSet and newCurSetr. A candidate is found if curSet has a point from every group. A result is inserted into P Q after checking for duplicates in steps (26-33). A duplicate check is done by a sequential match with the results in P Q. For a large value of k, a method similar to Algorithm 2 can be used for a duplicate check. If a new result gets inserted into P Q, then the value of rk is updated in step 18.

APPROXIMATE SEARCH: ProMiSH-A

In general, ProMiSH-A is more space and time efficient than ProMiSH-E, and is able to obtain near optimal results. The index structure of ProMiSH-A are similar to ProMISH-E.

The search technique of ProMiSH-A differs from ProMiSH-E in the initialization of priority queue PQ and the termination condition. ProMiSH-A starts with an empty priority queue PQ, unlike ProMiSH-E whose priority queue is initialized with k entries. ProMiSH-A checks for a termination condition after fully exploring a hash table at a given scale.

It terminates if it has k entries in its priority queue PQ. Since each point is hashed only once into a hash table of ProMiSH-A, it does not perform a subset duplicate check or a result duplicate check.

IV. IMPLEMENTATION AND ITS METHODOLOGY

Implementation is the stage where the theoretical design is converted into programmatically manner. In this stage we will divide the application into a number of modules and then coded for deployment. We have implemented the proposed concept on Java programming language with JEE as the chosen language in order to show the performance this proposed application. The front end of the application takes JSP, HTML and Java Beans and as a Back-End Data base we took My SQL data base along with a Some meaningful text files as data sets. The application is divided mainly into following 3 modules. They are as follows:

1. The Index Structure For Exact Search(ProMiSH-E) Module
2. The Exact Search Algorithm Module
3. Approximate Algorithm (ProMiSH-A) Module

1. THE INDEX STRUCTURE FOR EXACT SEARCH (PROMISH-E) MODULE

In This Project we start with the index for exact ProMiSH (ProMiSH-E). This index consists of two main components.

1. Inverted Index Ikp
2. **Hash table-Inverted Index Pairs HI**

The second component consists of multiple hash tables and inverted indexes referred to as HI. HI is controlled by three parameters: (1) Index level, (2) Number of random unit vectors, and (3) hash table size. All the three parameters are non-negative integers. These three parameters control the construction of HI.

2. THE EXACT SEARCH ALGORITHM MODULE

In exact search we are using ProMiSH-E algorithm that finds exact search results for NKS queries. In this algorithm hash indexes are used to store keys related to keywords. When user searching for file by entering search key it will compare that keyword with hash table keys and exact file will be retrieved by using following algorithms.

3. APPROXIMATE ALGORITHM (PROMISH-A) MODULE

In general, ProMiSH-A is more space and time efficient than ProMiSH-E, and is able to obtain near optimal results. The index structure of ProMiSH-A are similar to ProMish-E.The search technique of ProMiSH-A differs from ProMiSH-E in the initialization of priority queue PQ and the termination condition.

ProMiSH-A starts with an empty priority queue PQ, unlike ProMiSH-E whose priority queue is initialized with k entries. ProMiSH-A checks for a termination condition after fully exploring a hash table at a given scale. It terminates if it has k entries in its priority queue PQ. Since each point is hashed only once into a hash table of ProMiSH-A, it does not perform a subset duplicate check or a result duplicate check.

V. CONCLUSION

In this proposed thesis we finally came with a new method of finding the top-k nearest keyword set search in multi-dimensional datasets, where each and every data point is represented with a set of keywords. In this proposed thesis, we mainly designed and analyzed a novel index called ProMiSH based on random projections and hashing. Based on this index, we developed ProMiSH-E that finds an optimal subset of points and ProMiSH-A which searches near-optimal results with better efficiency. By conducting various experiments on our proposed method, our simulation results clearly tells that our proposed ProMiSH method is much fast than state-of-the-art tree-based techniques, with multiple orders of magnitude performance improvement.

VI. REFERENCES

[1] Pyle, D., 1999. Data Preparation for Data Mining. Morgan Kaufmann Publishers, Los Altos, California.

[2] A. Strehl and J. Ghosh, “Cluster ensembles: A knowledge reuse framework for combining multiple partitions,” J. Mach. Learning Res., vol. 3, pp. 583–617, 2002.

[3] G. Salton and C. Buckley, “Term weighting approaches in automatic text retrieval,” Inf. Process. Manage., vol. 24, no. 5, pp. 513–523, 1988.

[4] Hobbs, Jerry R.; Walker, Donald E.; Amsler, Robert A. (1982). "Proceedings of the 9th conference on Computational linguistics". **1**: 127–32. doi:10.3115/991813.991833.

[5] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, "Data Preprocessing for Supervised Learning", International Journal of Computer Science, 2006, Vol 1 N. 2, pp 111–117.

[6] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. McArthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz, “The expanding digital universe: A forecast of worldwide information growth through 2010,” Inf. Data, vol. 1, pp. 1–21, 2007.

[7] B. S. Everitt, S. Landau, and M. Leese, Cluster Analysis. London, U.K.: Arnold, 2001.

[8] A. K. Jain and R. C. Dubes, Algorithms for Clustering Data. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[9] "Unstructured Data and the 80 Percent Rule". Breakthrough Analysis. Retrieved 2015-02-23.

[10] D. Blei and J. Lafferty, “Correlated topic models,” Adv. Neural Inf. Process. Syst., vol. 18, pp. 147–154, 2006.

[11] Clustering document trees based on similarity measure by SM PRAJNA BODAPATI, A MANASA SUDHA in Asian journal of computer science and information technology 2012.

[12] Document Clustering Technique based on Noun Hypernyms By SM Bodapati Prajna in International Conference on advances Computer Science,Communication &amp; Bio 2011.

[13]Model based Clustering by S B.Prajna in National Conference on upcoming trends in IT.

VII. ABOUT THE AUTHORS

ASMA BEGUM is currently pursuing her Second Years M.Tech in Computer Science and Technology with Specialization in Artificial Intelligence and Robotics at Computer Science and System Engineering department, Andhra University College of Engineering, Visakhapatnam, Andhra Pradesh, India. Her area of interests includes Data Mining.

Prof. B.PRAJNA is currently working as a Professor in Computer Science and System Engineering Department, Andhra University College of Engineering, Visakhapatnam, Andhra Pradesh, India. She has more than 15 years of teaching experience. Her research interest includes Data Mining.