# CLINCH: Clustering Incomplete High-Dimensional Data for Data Mining Application*

Zunping Cheng[1], Ding Zhou[2], Chen Wang[1], Jiankui Guo[1], Wei Wang[1], Baokang Ding[1], and Baile Shi[1]

[1] Fudan University, China
{czp, chenwang, gjk, weiwang1, bkding, bshi}@fudan.edu.cn
[2] Pennsylvania State University, USA
dzhou@cse.psu.edu

**Abstract.** Clustering is a common technique in data mining to discover hidden patterns from massive datasets. With the development of privacy-maintaining data mining application, clustering incomplete high-dimensional data has becoming more and more useful. Motivated by these limits, we develop a novel algorithm **CLINCH**, which could produce fine clusters on incomplete high-dimensional data space. To handle missing attributes, CLINCH employs a prediction method that can be more precise than traditional techniques. On the other hand, we also introduce an efficient way in which dimensions are processed one by one to attack the "curse of dimensionality". Experiments show that our algorithm not only outperforms many existing high-dimensional clustering algorithms in scalability and efficiency, but also produces precise results.

**Keywords:** Clustering, Incomplete Data, High-Dimensional Data.

## 1 Introduction

The clustering is a primary technique in discovering hidden patterns from massive datasets. Common applications of cluster analysis involve scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, marketing, computation biology, and many others. [14]

Explosive progress in networking, storage, and processor technologies has led to deal with ultra large databases in data mining, which record tremendous, high-dimensional and transactional information. In tandem with this trend, concerns about informational privacy in data mining have emerged globally, for Data mining, with its promise to efficiently discover valuable, non-obvious information from large databases, is particularly vulnerable to misuse [5]. Specifically, a

* This paper was supported by the Key Program of National Natural Science Foundation of China (No. 69933010 and 60303008) and China National 863 High-Tech Projects (No. 2002AA4Z3430 and 2002AA231041).

Y. Zhang et al. (Eds.): APWeb 2005, LNCS 3399, pp. 88–99, 2005.
© Springer-Verlag Berlin Heidelberg 2005

person: 1) may not divulge at all the values of certain fields; 2) may not mind giving true values of certain fields; 3) may be willing to give not true values but modified values of certain fields [5]. All these situations will lead to the creation of missing attributes, i.e. privacy concerns.

Available studies in this mostly focus on how to fill the missing attributes with appropriate values [3, 8, 16]. Given a point $X_i$ with missing value in $j^{th}$ dimension, a general approach is to find the $K$-Nearest Neighbors *(KNN)* of $X_i$ and replace the missing attribute $X_{ij}$ with the average of the $j^{th}$ values of its *KNN* [16]. However, when application is extended to deal with high-dimensional data, this method becomes unreliable in that high-dimensional data are sparse-prone, which makes the *KNN* search on it meaningless [14, 18].

Algorithms to deal with the incomplete high-dimensional data are relatively few. Moreover, these algorithms are mostly mathematic-prone. Viewing from the perspective of data mining application, little work has started. Mining on the incomplete data would be a more and more popular issue.

In this paper, we propose an efficient method to cluster on incomplete high-dimensional data. We employ the three-step framework, just the same as most grid-based algorithm [10, 1, 9, 7], in our clustering algorithm. The problem is mainly solved by identifying the dense units. Our approach identifies the units by processing dimension by dimension.

There are two contributions from this method: I) lots of points are pruned after several dimensions, therefore the total cost is cut down; II) we propose an innovative prediction mechanism to solve the problem of incomplete data. From experiments and complexity analysis, we can see that our approach outperforms many existing approaches while maintaining a reasonable precision.

The remainder of this paper is organized as follows. In Section 2, we state the clustering problem. In Section 3, Algorithm *CLIQUE* is introduced simply. Algorithm *CLINCH* is developed in Section 4. In Section 5, we evaluate the performance on synthetic and real datasets via comparing *CLINCH* with *CLIQUE* and *FilV-DBScan*. Related work is presented in Section 6. Section 7 concludes the paper.

## 2 Problem Statement

**Definition 1 (DataSpace)**

*Given a set of $K$-dimensional data points $\{\overrightarrow{D_i}|i = 1, ..., n\}$, we have a $K$-by-$n$ data matrix $D$: $D = \{\overrightarrow{D_1}, \overrightarrow{D_2}, ..., \overrightarrow{D_n}\}$, where $\overrightarrow{D_i} = \{d_{1i}, d_{2i}, ...d_{Ki}\}^T$. In a normalized incomplete data space $D$, $d_{ij}$ could either be in $[0, 1]$ or* "uncertain". *The* "uncertain" *indicates the missing of this attribute at this point.*

**Definition 2 (Dimension)**

*Let a row of the data matrix $D$ be $Dim_j = \{d_{j1}, d_{j2}, ..., d_{jn}\}$. We refer to $Dim_j$ as the $j^{th}$ dimension of a data space. Apparently, a dimension is the set of $j^{th}$ attribute of all the points.*

**Definition 3 (Incompleteness)**

*Given a point* $\overrightarrow{D_i}$*, it has missing values if one or more of its attributes are uncertain. For example,* $\overrightarrow{D_i}$ *is a record in a* 5*-dimensional data space,* $\overrightarrow{D_i} = \{0.25, uncertain, 0.15, uncertain, 0.20\}$*, which means that record* $\overrightarrow{D_i}$ *has missing values in* $2^{nd}$ *and* $4^th$ *dimensions. Accordingly, the completeness* $\xi_j$ *of* $j^th$ *the dimension follows:*

$$\xi_j = \sum_{t=1}^{n} |\{d_{ji} | d_{ji} \neq uncertain, d_{ji} \in \overrightarrow{D_t}, i = 1, 2, ...n\}|.$$

*Generally speaking, if* $MAX(\xi_1, \xi_2, ..., \xi_m) \leq MININCOMPLETE$*, the dataset is complete. Here, MININCOMPLETE is a threshold specified by users and can be used to obtain an algorithmic gain.*

**Definition 4 (Unit)**

*Given a data space* $D$*, each unit* $u$ *is the intersection of one interval from each dimension. Formally,* $u = \{[l_1, h_1), [l_2, h_2), ..., [l_k, h_k)\}$*, where* $[l_i, h_i)$ *denotes the interval on the* $i^{th}$ *dimension. A unit is a dense unit if the points in this unit exceed a given support* $\epsilon$*. Namely, we say a unit is dense under support of* $\epsilon$*.*

**Definition 5 (Cluster)**

*Given a set of unit, a cluster is the maximal of connected dense units. Two points are in the same cluster if the units they belong to are connected or there exist a set of dense units that are each other connected.*

**Problem Statement:** Given a set of data points $D$, desired number of intervals $\sigma$ on each dimension and support $\epsilon$, the problem is how to predict which units the points with missing value belong to and generate clusters.

## 3 Algorithm CLIQUE

In [1], *CLIQUE*, an automatic subspace clustering algorithm for high-dimensional data, was proposed. *CLIQUE* consists of mainly three steps: 1) searches for the dense units; 2) clusters are generated by grouping the connected dense units; 3) concise descriptions for each cluster are produced based on minimal cluster descriptions *MDL*.

Among these three phases, the only phase that accesses database is the dense unit generation. Accordingly, the step1 takes up most of the time spent in discovering clusters. Remember that in *CLIQUE*, $k$-dimensional dense unit candidates are generated by self-joining all the $k-1$-dimensional dense units. However, given any several dimensions of the whole high-dimensional dataspace, there would be over-numbered points in every unit. This produces too many dense units in the first several dimensions in *CLIQUE*. Although most of these units will be pruned as the dimension goes high, to self-join on these large amount of units in the first several dimensions, however, would square the overall runtime. This deteriorates *CLIQUE*'s performance on high-dimensional data.

To clarify, let's take a look at a simple example. Given a $k$-dimensional data space $D$, *CLIQUE* seeks the $k$-dimensional dense units level by level. Thus while generating all the 3-dimensional dense units, CILQUE self-joins the dense units in all 2-dimensional subspaces. Now let's estimate how large a number of dense units there will be if $D$ is a high-dimensional database: in all, there are $C_k^2 = k \times (k-1)$ 2-dimensional subspaces. On every subspace, there will be $(I) \times (I)$ units, where $(I)$ is the number of intervals on each dimension. Note when $D$ are projected to any 2-dimensional subspaces, there will be over-numbered points in every units. This probably produces as many as $k \times (k-1) \times (I) \times (I)$ 2-dimensional dense units in all. Moreover, when such large number of units are self-joined, the cost time is squared.

## 4 Algorithm CLINCH

In this section, we introduce our approach *CLINCH* (i.e. **CL**ustering **INC**omplete **H**igh-dimensional data). We employ the framework like most grid-based clustering algorithms. Figure 1 illustrates the steps of algorithm *CLINCH*.

| | |
|---|---|
| **Input:** | Dataset $D$, support threshold *minsup*; |
| **Output:** | A set of cluster's IDs; |
| **Method:** | CLINCH($D$, *minsup*) |
| 1: | DenseUnitGen($D$,*minsup*) //generate the dense units; |
| 2: | Search for clusters with dense units set; |

**Fig. 1.** Algorithm CLINCH

In the first step, we partition the space into several equi-sized units[1]. Then the units with points more than given support are marked as the dense units.

In the second step, the connected dense units are tagged the same ID as clusters. Two dense units are named to be "connected" if they share a common face [1]. Given a graph $G$, let every dense unit be a vertex in $G$. There is an edge between two dense units if and only if the two corresponding dense units are "connected". Then the problem of step 2 is equivalent to searching for the connected components in a graph. In implementation, we use depth-first search algorithm [2] in discovering these components. Since the problem is projected into the search of the graph, the clustering result will not be affected by the order of the records, ie. CLINCH is stable.

We propose to process the whole data space dimension by dimension. The phrase "dimension by dimension" means that 1) firstly dense units in *K-1-d* space are produced and then 2) dense units on *K-d* space are produced by combining

[1] In some paper, the interval assignments of partitioning of units are studied. Here we assume the space is partitioned evenly.

**Input:** Dataset $D$, support threshold $minsup$;
**Output:** A set of dense units;
**Method:** DenseUnitGen($D$, $minsup$)
1: $DS \leftarrow$ determine and sort the order of $D's$ dimensions;
2: $UList \leftarrow$ FullDim($DS$,$minsup$) //recognition on full dimensions;
3: IncompleteDim($DS$,$minsup$,$UList$,$DimID$) //recognition on incomplete dimensions;

**Fig. 2.** Procedure DenseUnitGen

the mining results on *K-1-d* and dense intervals on the *Kth* dimension. At first, we sort all dimensions. Then we search for the dense units in the first several complete dimensions. Suppose there are $\beta$ complete dimensions in all, we produce $\beta - dimensional$ dense units and pass them to the following step. Then the rest dimensions are processed with the received $\beta - dimensional$ dense units. At last a set of dense units will be generated in whole dimensions. Figure 2 shows the procedure for generating dense units.

### 4.1 Determine the Order of Dimension for Process

Intuitively, we firstly deal with the complete dimensions and then the incomplete ones. We handle the complete dimensions in the decreasing order of their entropies [7]. The entropy of a dimension is defined in [7, 10]. Since lager entropies mean more information, if we deal with dimensions with larger entropies, the prediction on the missing attributes will be more precise. After finishing all the complete dimensions, characteristics of clusters on this complete subspace are built through entropies, and will be employed for the prediction of missing attributes through the following incomplete dimensions. This process is similar to building a decision-tree for the first several dimensions, which would be used in later prediction [5]. Intuitively, we optimize the grouping of points for prediction if we follow the decreasing entropies. Similar idea has already been widely accepted in the study of classification algorithms [7].

On the other hand, we process the incomplete dimensions after all the complete ones are processed. For these incomplete dimensions, we handle them according to their completeness $\xi_j$. This is to guarantee that dimensions with more uncertain records are processed later, which maximizes the information used for prediction.

After the order of dimension is determined, we proceed to recognize the dense units.

### 4.2 Recognition on Full Dimensions

We benefit from the monotonicity introduced in [1]. If a unit is dense in its $k$-dimensional space, then it will be also dense in any of its $k-1$-dimensional space [1]. With this feature, we could prune many units in the first several

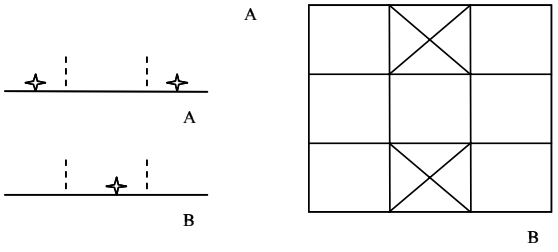


**Fig. 3.** Generation of 2-d candidates from 1-d dense units

**Input:** Dataset sorted by entropy of Dimension $DS$, support threshold $minsup$;
**Output:** a set of $\beta - dimensional$ units
**Method:** FullDim($DS$, $minsup$)

```
1:  K ← number of complete dimensions;
2:  C_0 ← 1-dimensional dense units on DS_0;
3:  C[1, ...K − 1] ← NULL;
4:  for i from 1 to K − 1
5:      U_i ← Generate the 1 − dimensional dense units on the dimension of DS[i];
6:      C_i ← C_{i−1} × U_i; //generate the i-dim unit candidates from C_{i−1} and C_i;
7:      scan subspace from DS_0 to DS_i and keep the dense units in C_i;
```

**Fig. 4.** Procedure FullDim

dimensions if we recognize the dense units dimension by dimension. Our process is illustrated in Figure 3: let A be the first dimension and B be the second. Figure 4 shows the pseudo-code of our approach.

In line 6 of Figure 4, $i$-dimensional unit candidates are generated from the $1-dimensional$ dense unit $U_i$ and $C_{i-1}$. Thus the time cost of this step is bounded by the number of element in each dimension. Given every $k-dimensional$ dense unit is also dense in any of its $k-1$ subspace, our $k-1$ dense unit list $C_{k-1}$ contains the dense units in $k-dimensional$. Then by also checking the ones on the $i^th$ dimension, we further limit the number of candidates for following prune. Line 7 then check on the subspace to maintain the truly dense one, whose implementation is straightforward.

Let's go back to the example in Section 2. Compared with the self-join of CLIQUE, the join of CLINCH only generate as many as $k \times (k-1) \times (I_{max}) \times (I_{max})$ dense units. Here, $I_{max} = MAX(\{I_i | i = 1, ..., k)\}$, $I_i$ is the amount of dense unites of the $i^{th}$ dimension. Since $I_{max} \leq I$, it is clear that the join of CLINCH can get less cost than the self-join of CLIQUE. Furthermore, the employment of the dimension-by-dimension processing in the dense unit recognition not only ease the efficiency in the self-join step in *CLIQUE* but also provides the possibility to cluster incomplete data without filling the value beforehand.

### 4.3 Recognition on Incomplete Dimensions

After all the complete dimensions are processed, we pass the dense units to the step 3. Our idea is to employ the information from the processed several dimensions to predict those missing values. Based on the $dimension-by-dimension$ approach, we introduce a decision-tree like mechanism to enable predication for incomplete points. An example of such prediction is illustrated in Figure 5. Given four $3-dimensional$ points $A(0.4, 0.4, X)$, $B(0.5, 0.4, 0.8)$, $C(0.85, 0.45, 0.2)$ and $D(0.7, 0.5, 0.3)$, assume they are all complete in their first 2 dimension and their positions are illustrated as follow:

This algorithm is illustrated in Figure 6.

### 4.4 Time Complexity

The time complexity of Algorithm *CLINCH* mainly consists of three parts:

1. Sort the dimensions by their entropy.
2. Generate $1-dimensional$ dense unit.
3. Generate $k-dimensional$ units dimension by dimension.

Among these three steps, part 1 and 2 cost $O(k*n)$ and $O(n)$ respectively. On the other hand, the time complexity of part 3 is bounded by $O(\sum_{1}^{k-1} T_i)$, where $T_i$ denotes the time in dealing with $n$ points on $i^{th}$ dimension. When it is complete in dimension $i$, time to handle $n$ points is $O(n)$, otherwise the time depends on the completeness $\xi_i$. Formally,

$$T_i = \begin{cases} O(n) & The\ i^{th}\ dimension\ is\ complete \\ O(n*(1-\xi_i) + \sum_{1}^{n*\xi_i} a_j) & Otherwise \end{cases}$$

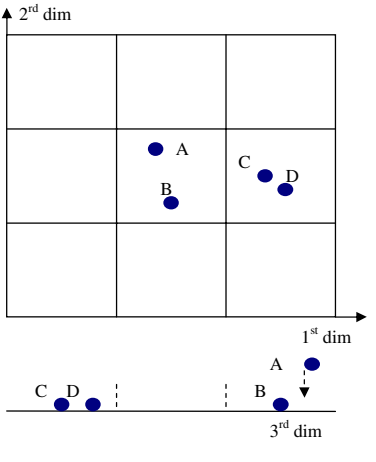


**Fig. 5.** Example for 3-dimensional units recognition

**Input:** Dataset sorted by entropy of Dimension $DS$, support threshold $minsup$, a set of $\beta - dimensional$ units $UList$, Dimension ID $DimID$;
**Method:** IncompleteDim($DS$, $minsup$, $UList$, $DimID$)

```
1:  C_(DimID-1) ← UList;
2:  for i from DimID to DS.Dim − 1
3:      U_i ← Generate the 1 − dimensional dense Unit on the dimension of DS[i];
4:      C_i ← C_(i−1) ∪ U_i
5:      for every point P in the subspace form D[0...i]
6:          if P is complete in i^th dimension then
7:              determine which unit candidate P belongs to
8:          else
9:              find P's nearest complete neighbor Pnbr from S
10:             //S is the i − 1-dimensional unit that Pnbr belongs to
11:             record P and its Pnbr;
12:     include each incomplete point P to the i-dim unit Pnbr belongs to;
13:     remove the non-dense unit in C_i;
```

**Fig. 6.** Procedure IncompleteDim

Where $a_j$ is the largest size of unit that an incomplete point needs to seek in for neighbor.

Overall, the time complexity is $O(k * n + \sum_{1}^{k-1} T_i + n)$ , which is bounded by $O(k * n)$. This much outperforms the exponential time of CILQUE.

## 5 Experiments

In this section, we evaluate the performance of *CLINCH*. Criterion in estimating the performance of *CLINCH* includes its efficiency and quality of clustering results. In efficiency, we record the total CPU time for comparison with *CLIQUE* and the way introduced in [16] followed by *DBSCAN*. The efficiency was tested on both synthetic and real data. The synthetic data generator was introduced in [22]. The real dataset is from the completely specified database *MUSK2* from *UCI*[2] machine learning repository. We evaluate the precision in the way introduced in [21], which would be covered in details later. We implemented all the algorithms in C++. The experiments have been run on Pentium IV with 2.2 GHz containing 512MB DDR of main memory and Windows XP as operating system.

### 5.1 Efficiency

We use real data as well as synthetic data in the experiments. The synthetic data generator is introduced in [22]. We could set the number and size of clusters,

[2] http://www.cs.uci.edu/ mlearn

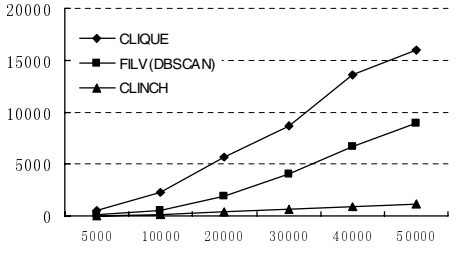


**Fig. 7.** Time vs. Dataset Size (50-d)

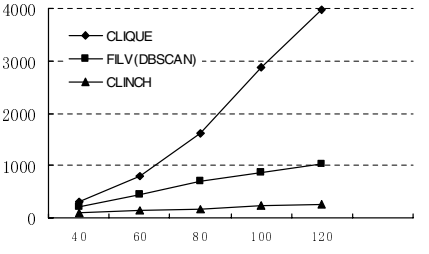


**Fig. 8.** Time vs. Dim (size of 10000)

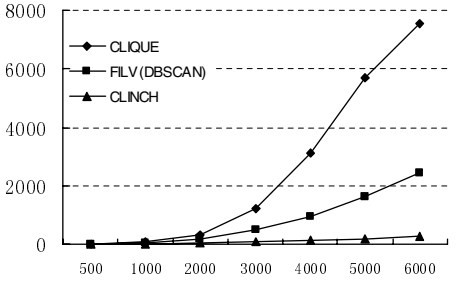


**Fig. 9.** Time vs. Dataset Size on MUSK2

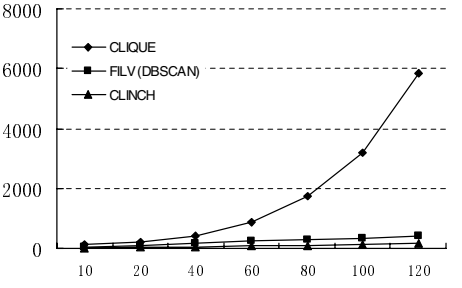


**Fig. 10.** Time vs. Dim on MUSK2

size and dimensionality of dataset etc. To test the performance of *CLINCH* on different size of real data, we extract a series of subsets of *MUSK2*. Experiments on the efficiency of *CLINCH* include its sensitivity to the dimensionality and size of dataset.

Using cluster generator, we set to generate 5 clusters of 50-dimensional database in different sizes. We could see from the figure 7 that *CLINCH* scales well with the increase of database size. Besides, *CLINCH*'s scalability with dimension was also investigated. We tested three algorithms on datasets with 10000 points. The effects of dimensions are shown in figure 8.

We also conducted the comparison on real dataset. The dataset is from the machine learning repository *MUSK2* from *UCI*. Scalability with dimension and database size is tested by retrieving either the subspaces of dimensions and subsets of data. Figure 9 and 10 show performances of *CLINCH* on different database sizes and dimensionality.

### 5.2 Precision

In this section we would study what precision the *CLINCH* maintains and could see that *CLINCH* produces the same clustering results if both performed on complete data.

We employ the dataset from completely specified *UCI* dataset *MUSK2* for experiments. We generate the incomplete data space by randomly removing some attributes of points.

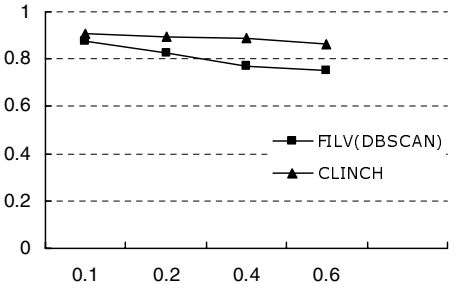


**Fig. 11.** Precision vs. Missing Dim

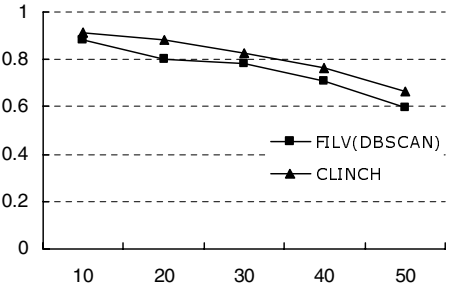


**Fig. 12.** Precision vs. Missing Percent

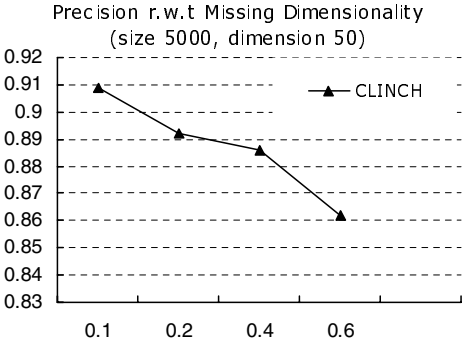


**Fig. 13.** Clinch Precision (compared with CLIQUE)

In the sea of filling-value algorithms for incomplete data, we choose the approach introduced in [16], which is simple and easy to implement. In assessing the quality of *CLINCH*, we benefit from the method introduced in [21]. Specifically, we measure the precision by solving the label-matching problem. We here compared our results with *CLIQUE*, which was taken as producing the acceptable grouping of points.

Figure 11 illustrates the precision with respect to the dimensionality that miss values.

Firstly, we could see from Figure 13 that the CLINCH produces very similar clustering results as CLIQUE. In figure 11, we set the missing percentage of every dimension to be a random value no larger than 0.5, we presume that dimension with a large missing percentage will be useless. Figure 12 show the effect of missing percentage. We randomly miss a certain percentage of the attributes in some dimensions, precision on which is illustrated above.

## 6 Related Work

Several approaches are proposed to fight the problem of clustering high-dimensional data. The first category is to perform dimensionality reduction

before clustering: points are projected to lower dimensional space to ease the traditional on reduced data space [4, 6]; Another category in attacking the high-dimensional clustering is based on grid partitioning [1, 9, 7, 20]. Algorithms in this category first divide the space into rectangular units and keep the high-density ones. Then the high-density units are combined to their neighbors to form clusters and summaries.

Comparatively, in handling the points with missing attributes. Much work has been done on filling the missing attributes with appropriate values. Several simple, inexpensive and easy to implement techniques were introduced in [16]. Besides, imputation, statistical or regression procedures are widely used in estimating the missing values [13, 15]. Similar idea by reconstruction is also illustrated in [3]. However, these techniques are prone to estimation errors when the dimensionality increases. There are also some approaches that view this problem from a different angel: they extract principal components without elimination or imputation [11, 19]. Similar PCA-like methods for missing value estimation include [11, 17].

## 7 Conclusion

In this paper, we develop an effective and efficient method to clustering on incomplete high-dimensional data. Our two contributions include contributing a fast high-dimensional clustering technique as well as the proposal of an prediction technique based on it. Our experiments and theoretical proof show our algorithm CLINCH has a good performance both in efficiency and precision of clustering incomplete data. In the future, we would like to transplant the algorithm to other fields such as multimedia mining and pattern recognition. Meanwhile, to meet the need of documents classification, further study in super high-dimensional data is necessary.

## References

1. R. Agrawal, J. Gehrke, D. Gunopulos,and P. Raghavan. Automatic subspace clustering of high-dimensonal data for data mining applications. In *proc of the ACM SIGMOD Conference*, 94-105, Seattle, WA, 1998.
2. A. Abo, J. Hopcroft and J. Ullman. The Design and Analysis of Computer Algorithms. Addison-Welsley, 1974..
3. C.C. Aggarwal and S. Parthasarathy. Mining Massively Incomplete Data Sets by Conceptual Reconstruction. In *proc of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.
4. C.C. Aggarwal, C. Procopius,J.L. Wolf, P.S. Yu and J.S. Park. Fast Algorithm for Projected Clustering. In *proc of the ACM SIGMOD Conference*, 61-72, Philadelphia, PA, 1999.
5. R. Agrawal and R. Srikant. Privacy Preserving Data Mining. In *ACM SIGMOD*, 2000.
6. C.C. Aggarwal and P.S. Yu. Finding generalized projected clusters in high dimensional spaces. Sigmod Record, 29,2,70-92, 2000.

7. C. Cheng, A. Fu and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *proc of the 5th ACM SIGKDD Conference*. 84-93, San Diego, CA, 1999.
8. Z. Ghahramani and M. I. Jordan. Learning from incomplete data. *Department of Brain and Cognitive Sciences, Paper No. 108*, MIT, 1994.
9. S. Goil,H. Nagesh and A. Choudhary. MAFIA: Efficient and scalable subspace clustering for very large data sets. Technicial Report CPDC-TR-9906-010, Northwestern University, 1999.
10. J.W. Han. etc. Data Mining: Concepts and Techniques. Morgan Kaufmann Press, June, 2000.
11. K. Honda, A. Yamakawa, A. Kanda and H. Ichihashi. An application of fuzzy c-Means Clustering to PCA-Like method for missing value estimation. In *Proc. 16th Int. Conf. on Production Research*, Prague, Czech, July 2001.
12. I. Joliffe Principal Component Analysis. Springer-Verlag, New York, NY, 1986.
13. R. Little and D. Rubin. Statistical Analysis with Missing Data Values. Wiley Series in Prob. and Stat,1987.
14. P. Berkhin. Survey of Clustering Data Mining Techniques. In: *Accrue Sotware*, 2002.
15. J.R. Quinlan. Programs for Machine Learning. Morgan Kaufman, 1993.
16. J. Rodas and J. Gramajo. Classification and Clustering Study in Incomplete Data Domain. *Informatic Systems and Languages Department, Technical University of Catalonia.*, 2000.
17. T. Shibayama. A PCA-Like Method for Multivariate Data with Missing Values. Japanese Journal of Educational Psychology, Vol. 40, 257-265, 1992.
18. M. Steinbach, L.Ertöz and V. Kumnar. The Challenges of Clustering High Dimensional Data. In *Applications in Econophysics, Bioinformatics, and Pattern Recognition.*
19. H. Shum,K. Ikeuchi and R. Reddy. Principal Component Analysis with Missing Data and its Application to Polyhedral Object Modeling. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol 17, No. 9, 854-867, 1995.
20. D. Zhou, Z.P. Cheng,C. Wang,H.F. Zhou, W, Wang and B.L. Shi. SUDEPHIC: Self-tuning Density-based Partitioning and Hierarchical Clustering In *proc of the 9th International Conference on Database Systems for Advanced Applications*, Jeju Island, Korea, 2004.
21. H.Y. Zha,C. Ding, M. Gu, X.F. He and H. Simon. Spectral Relaxation for K-means Clustering. Neural Info. Processing Systems NIPS 2001.
22. M. Zait and H. Messatfa. A comparative study of clustering methods. Future Generation Computer Systems, 13(2-3): 149-159, November 1997.