Efficiently Detecting Arbitrary Shaped Clusters in Very Large Datasets with High Dimensions *

Dantong Yu, Surojit Chatterjee, Gholamhosein Sheikholeslami and Aidong Zhang
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260 USA

Abstract

Multimedia databases typically contain data with very high dimensions. Finding interesting patterns in these databases poses a very challenging problem because of the scalability, lack of domain knowledge and complex structures of the embedded clusters. High dimensionality adds severely to the scalability problem. It has been shown that the wavelet-based clustering technique, WaveCluster, is very efficient and effective in detecting arbitrary shape clusters and eliminating noisy data for low dimensional data. In this paper, we introduce *HiperWave*, an approach to applying wavelet-based techniques for clustering high dimensional data. Using a hash-based data structure, our approach makes intelligent use of available resources to discover clusters in the dataset. We demonstrate that the cost of clustering can be reduced dramatically yet maintaining all the advantages of wavelet-based clustering. This hash-based data representation can be applied for any grid-based clustering approaches. Finally, we introduce a quantitative metric to measure the quality of the resulting clusters. The experimental results show both effectiveness and efficiency of our method on high dimensional datasets.

1 Introduction

In multimedia databases, objects such as video, audio streams, and images. are mapped to feature vectors in high-dimensional feature spaces. There are many other databases such as financial, crystallography and corporate databases, where very large high dimensional datasets with numerical attributes exist. Clustering in data mining is the discovery of interesting patterns that may exist in the underlying data. In this paper we explore a *data clustering* method in the high dimensional data mining problem.

Because of the large size of these databases, a primary requirement for clustering algorithms for data mining is efficiency. The high dimensionality adds the problem of scalability to this. The clustering technique should be fast and scalable with the number of dimensions and the size of the input. Also, due to the diverse nature and characteristic of the source of the data, the clusters may assume arbitrary shapes. They may be nested within one another, may have holes inside, or may possess concave shapes. The problem of handling arbitrary shapes in high dimensions is

* This research is partially supported by an NSF CAREER grant IIS-9733730 and a grant from Xerox Corporation.

particularly complex. A good clustering approach should be unaffected by outliers (noise) and should detect them effectively.

The diverse methods of extracting feature vectors and their representation makes it impossible for the user to obtain any knowledge about the number of clusters present. Thus clustering algorithms should assume minimum domain knowledge, e.g., number of clusters or underlying probability distribution. Also, clustering algorithms should be insensitive to the order of the input data. As discussed in [10], another desirable property for clustering algorithms should be the ability to produce clusters at different levels of granularity which is termed as *multiresolution* property. This is particularly suitable for multimedia databases where objects may be needed to be classified at different scales and levels of details. An interested user should have the flexibility to explore each of the clusters at different levels of detail. This hierarchical level of clustering will make indexing and subsequent retrieval of high dimensional data more efficient.

1.1 Related Work

One category of clustering methods is partitioning algorithms. Partitioning algorithms construct a partition of a database of N objects into a set of K clusters. Usually they start with an initial partition and then use an iterative control strategy to optimize an objective function. There are mainly two approaches i) k-means algorithm, where each cluster is represented by the center of gravity of the cluster, ii) k-medoid algorithm, where each cluster is represented by one of the objects of the cluster located near the center. Ng and Han introduced CLARANS (Clustering Large Applications based on RANdomaized Search) which is an improved k-medoid method [9]. This is the first method that introduces clustering techniques into spatial data mining problems. Compared to the previous k-medoid algorithms, CLARANS is more effective and efficient.

The other category of clustering methods includes hierarchical algorithms which create a hierarchical decomposition of the database. The hierarchical decomposition can be represented as a *dendrogram* [4]. The algorithm iteratively splits the database into smaller subsets until some termination condition is satisfied. Hierarchical algorithms do not need K as an input parameter, which is an obvious advantage over partitioning algorithms. The disadvantage is that the termination condition has to be specified. BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) uses a hierarchical data structure called CF-tree which is a height balanced tree that stores the clustering features [16]. BIRCH tries to produce the best clusters with the available resources (memory and I/O time).

Ester et. al. [3] presented a clustering algorithm DBSCAN relying on a density-based notion of clusters. It is designed to discover clusters of arbitrary shapes. The key idea in DBSCAN is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points. DBSCAN can separate the noise (outliers) and discover clusters of arbitrary shapes. It uses R*-tree to achieve better performance. CURE (Clustering Using Representatives) utilizes multiple representative points for each cluster that are generated by selecting well scattered points from the cluster and then shrinking them toward the center of the cluster by a specified fraction [5].

Recently a number of algorithms were presented which quantize the space into a finite number of cells and then do all operations on the quantized space. The main characteristic of these approaches is their fast processing time which is typically independent of the number of data objects. They depend only on the number of cells in each dimension in the quantized space. Wang et al proposed a STatistical INformation Grid-based method (STING) for spatial data mining [14]. They divide the spatial area into rectangular cells using a hierarchical structure. They store the statistical parameters of each numerical feature of the objects within cells. In STING, the hierarchical representation of grid cells is used to search for queries or assign a new object to the clusters.

CLIQUE clustering algorithm identifies dense clusters in subspaces of maximum dimensionality [1]. It argues that finding the clusters in the subspaces of the original space is more effective, because many dimensions can have noise or uniformly distributed values. Also in high dimensional spaces, the average density of points anywhere in the data space is likely to be quite low. CLIQUE partitions the feature space into cells. To approximate the density of the data points, it counts the number of points in each cell. The clusters are unions of connected high density cells within a subspace. It generates cluster descriptions in the form of DNF expressions. Note that while CLIQUE focuses on finding clusters embedded in subspaces of high dimensional data, our proposed approach, similar to BIRCH and DBSCAN, is designed to detect clusters in full dimensional space. Hence, in that regard the scope and goal of our method are different from those of CLIQUE.

We proposed WaveCluster which partitions the feature space into cells and applies wavelet transform on them [10]. Using multiresolution property of wavelets, WaveCluster can detect arbitrary shape clusters at different degrees of accuracy. Though this approach meets all the desirable properties of a good clustering technique, its performance in terms of required memory and time degrades as the dimensionality of the data increases. Thus, it requires modifications to better handle high dimensional datasets.

1.2 Our Contributions

In this paper, we present an approach, named *HiperWave*[1], to efficiently clustering high dimensional data using wavelet transform. Our goal is to incorporate optimizations for achieving the time efficiency and also to propose strategies for memory management. Using a hash table data structure to represent the dataset, our approach makes intelligent use of available resources to efficiently perform clustering on high dimensional datasets. This hash-based data representation technique

[1]HiperWave is named after HIgh-dimensional PattErn Recongnizer using WAVElet.

can be applied for any grid-based clustering approaches. We demonstrate that the cost of clustering can be reduced significantly yet maintaining all the advantages of wavelet-based clustering. Finally, we introduce a new quantitative metric to measure the quality of the resulting clusters. We also present comprehensive experimental results to demonstrate the effectiveness and efficiency of our method on high dimensional datasets.

The rest of the paper is organized as follows. We first formally describe the problem to be addressed in Section 2. Section 3 presents the algorithm and data structure used followed by analysis of the time complexity given in Section 4. In Section 5, we present the experimental evaluation and effectiveness of this approach. Finally, in Section 6, concluding remarks are offered.

2 Problem Formalization

2.1 Definitions and Notation

Following the definition of Agrawal et. al. [1], let $A = A_1, A_2, \ldots, A_d$ be a set of bounded, totally ordered domains and $S = A_1 \times A_2 \times \ldots \times A_d$ be a d-dimensional numerical space or feature space. $A_1, \ldots, A_d$ are referred as dimensions of S. The input data set is a set of d-dimensional points $\mathbf{O} = \{\mathbf{o_1}, \mathbf{o_2}, \ldots, \mathbf{o_N}\}$, where $\mathbf{o_i} = \langle o_{i1}, o_{i2}, \ldots, o_{id} \rangle$, $1 \leq i \leq N$. The j-th component of $\mathbf{o_i}$ is drawn from domain A_j.

We first partition the original feature space into nonoverlapping hyper-rectangles which we call *cells*. The cells are obtained by segmenting every dimension A_i into m_i number of intervals. Each cell $\mathbf{c_i}$ is the intersection of one interval from each dimension. It has the form $\langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$ where $c_{ij} = [l_{ij}, h_{ij})$ is the right open interval in the partitioning of A_j. Each cell $\mathbf{c_i}$ has a list of statistical parameters $\mathbf{c_i} \cdot \mathbf{param}$ associated with it.

We say that a point $\mathbf{o_k} = \langle o_{k1}, \ldots, o_{kd} \rangle$ is contained in a cell $\mathbf{c_i}$, if $l_{ij} \leq o_{ki} < h_{ij}$ for $1 \leq j \leq d$. The list $\mathbf{c_i} \cdot \mathbf{param}$ keeps track of the statistical properties such as aggregation, mean, variance, and the probability distribution of the data points contained in the cell $\mathbf{c_i}$. In general, in grid-based approaches by a single pass through the dataset the containment relations are discovered and appropriate statistical parameters are computed. Each cell has information about the density of the data contained in the cell. Thus, the collection of $\mathbf{c_i} \cdot \mathbf{param}$ summarizes the dataset.

We choose the number of points contained in each cell as the only statistic to be used. That is, we use $\mathbf{c_i} \cdot \mathit{count}$ to be the $\mathbf{c_i} \cdot \mathbf{param}$. In our approach we apply wavelet transform on $\mathbf{c_i} \cdot \mathit{count}$ values. We define the *transformed space* as the set of cells after wavelet transformation on the count values of the cells in the quantized space.

In our approach we take an all point stand on defining clusters, i.e., we consider that all the points within a cluster are representatives of the cluster. We introduce the following definitions to be used for the rest of the paper.

Definition 1 *(Empty cell) A cell in the quantized space with 0 count value is called an empty cell.*

Definition 2 *(Nonempty cell) A cell in the quantized space with nonzero count value is called a nonempty cell.*

Definition 3 *(Significant cell) A cell is a* significant *cell if its* count *field*[2] *in the transformed space is above a certain threshold τ.*

Definition 4 *(ϵ-neighbor) A cell $\mathbf{c_1}$ is an ϵ-neighbor of cell $\mathbf{c_2}$ if either both are significant cells (in transformed space) or nonempty cells (in quantized space) and $D(\mathbf{c_1}, \mathbf{c_2}) \leq \epsilon$, where D is an appropriate distance metric and $\epsilon > 0$.*

We can extend the definition of ϵ-neighborhood to k-ϵ-neighborhood.

Definition 5 *(k-ϵ-neighbor) A cell $\mathbf{c_1}$ is a k-ϵ-neighbor of a cell $\mathbf{c_2}$ if both are significant cells (in transformed space) or both are nonempty cells (in quantized space) and if $\mathbf{c_1}$ is one of the k prespecified ϵ-neighbors of $\mathbf{c_2}$.*

Definition 6 *(k-connected) Two cells $\mathbf{c_1}$ and $\mathbf{c_2}$ are k-connected if there is a sequence of cells $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_j}$ such that $\mathbf{p_1} = \mathbf{c_1}$ and $\mathbf{p_j} = \mathbf{c_2}$ and $\mathbf{p_{i+1}}$ is a k-ϵ-neighbor of $\mathbf{p_i}$, $1 \leq i \leq j$.*

Definition 7 *(Cluster) A cluster $\mathcal{C}$ is a set of significant cells $\{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}\}$ which are k-connected in the transformed space.*

Given a set of N database points $\mathbf{O} = \{\mathbf{o_1}, \mathbf{o_2}, \ldots, \mathbf{o_N}\}$ in the d-dimensional feature space S, our goal is to find the clusters as defined above. After clustering, each cell in the feature space S will have a label indicating the cluster that it belongs to.

2.2 Wavelet-Based Clustering and New Problems

In [10] we showed how a signal processing approach can solve the clustering problem in data mining. In this method the collection of objects in the d-dimensional feature space compose a d-dimensional signal. The high frequency parts of the signal correspond to the regions of the feature space where there is a rapid change in the distribution of objects, which is boundaries of clusters. The low frequency parts of the d-dimensional signal which have high amplitude correspond to the areas of the feature space where the objects are concentrated. These are the clusters in the original space. This view of the feature space allows us to use signal processing techniques such as wavelet

[2] In case we keep more elaborate information about each cell then we can specify a range of values for each of the parameters of each cell for determining whether it is a significant cell.

transform to separate out different frequency bands in the signal. In [10] it has been shown that wavelet transform techniques are very effective and efficient for low dimensional data. This method can detect arbitrary shape clusters and has the multiresolution property. But this method runs into problem when number of dimensions is very high. For simplicity, let us assume that all the intervals in each of the d_i dimensions is same, i.e., $m_1 = m_2 = \ldots = m_d = m$. Also assume that each of the intervals has equal length. So there will be $K = m^d$ cells. So the space complexity of WaveCluster algorithm as proposed in [10] is exponential with respect to the dimensionality of the data. Let the total number of data points be N. It was also shown in [10] that the time complexity of the WaveCluster algorithm is determined by the bigger of $O(N)$ and $O(K) = m^d$. In WaveCluster we assumed that $N >> K$ which is a valid assumption in the case of low d. For high dimensions we will have $N << K$. So the running time will also grow exponentially with the dimensions of the data if WaveCluster is applied in high dimensions.

But in high dimensions it is expected that data will be sparse and most of the cells in the quantized space will be empty. An efficient way of storing only the nonempty cells in the quantized space is expected to drastically reduce the space complexity. In HiperWave, we propose a data structure to keep track of the nonempty cells only. But performing a convolution operation such as wavelet transform on this representation is a nontrivial problem. We propose an accumulative approach to calculate wavelet transform in high dimensional space. And finally we find k-connected components in high dimensional space by a depth-first search through the hash table.

3 High Dimensional Data Clustering

As discussed in the earlier sections, we observe that in the high dimensional feature space the actual dense areas are very small compared to the whole space. A hash table can be an effective method to exclusively deal with this kind of situation. In this section we present the algorithm and provide an indepth analysis of the data structure used, the problems associated in adapting this type of data structure and how we solve them.

3.1 Algorithm

Given a set of spatial objects $o_i, 1 \leq i \leq N$, the goal of the algorithm is to detect clusters and assign labels to the objects based on the cluster they belong to. We use wavelet transform as a tool for facilitating clustering. The main idea here is to efficiently represent high dimensional data in limited memory and perform wavelet transform as well as connected component analysis on this representation. The outline of the algorithm is given below.

Input: High dimensional data objects' feature vectors
Output: clustered objects

1. Quantize feature space, aggregate objects into the cells and construct a hash table H.
2. Apply wavelet transform on H and save the significant cells in a new hash table H'.
3. Find the connected components (clusters) in H'.
4. Remove clusters which have low number of cells.
5. Assign labels to the cells.
6. Make the lookup table.

3.2 Data Structure

In the quantized space, every cell $\mathbf{c_i}$ has the form of $\langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$, which is called the *key* or *index* for $\mathbf{c_i}$. The address of a cell in the hash table can be calculated by applying appropriate hash function on the index of a cell. A hash table requires much less storage than a direct-address table, which was used in [10]. Specifically, the storage requirement can be reduced from $O(m^d)$ to $\Theta(N' \times d)$, where N' is the number of nonempty cells in the quantized feature space.

After hashing, a cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$ is stored in the hash bucket $h(\mathbf{c_i})$; that is, a **hash function** h is used to compute the address for the cell $\mathbf{c_i}$. Formally, the hash function h maps the universe U of $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$ into the entries in the *hash table* $T[0 \ldots n-1]$, where n is the number of buckets in the hash table. That is, $h : U \to 0, 1, ..., n-1$.

Hash function. Here we define some operators to give a formal description of the hash function used.

- Operator $\bigoplus$ is defined to be bitwise exclusive OR. When the operands are one bit, this operator is a bit operator. When the two operands a and b are binary vectors with the same length, the result c will be a binary vector which has the same length as a and b. It has these properties: 1) $a \bigoplus a = 0$, and 2) $a \bigoplus 0 = a$.
- Unary operator $\#$ is defined as $\#(a) = a_1 \bigoplus a_2 \bigoplus ... \bigoplus a_\mu$, where $a_i \in \{0, 1\}$, $1 \leq i \leq \mu$, is the i-th bit of a and μ is the length of binary vector a.
- Binary operator $\wedge$ is same as applying bitwise *and* operation on two vectors.

We randomly generated an integer matrix $A_{l \times d}$, where $l = log_2 n$ and n is the number of hash table buckets as follows:

$$A_{l \times d} = \begin{pmatrix} r_{1,1} & r_{1,2} & \ldots & r_{1,d} \\ r_{2,1} & r_{2,2} & \ldots & r_{2,d} \\ \ldots & \ldots & \ldots & \ldots \\ r_{l,1} & r_{l,2} & \ldots & r_{l,d} \end{pmatrix}.$$

We assume that $r_{i,j}$ and c_{ij} are both bit vectors. Each of which is obtained by binary conversion of the corresponding integer values. So each of original vectors $\mathbf{c_i}$ is now represented by the set of bit vectors $\langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$. We define hash function $h(c_{i1}, c_{i2}, \ldots, c_{id})$ as follows:

$$h(c_{i1}, c_{i2}, \ldots, c_{id}) = \begin{pmatrix} r_{1,1} & r_{1,2} & \ldots & r_{1,d} \\ r_{2,1} & r_{2,2} & \ldots & r_{2,d} \\ \ldots & \ldots & \ldots & \ldots \\ r_{l,1} & r_{l,2} & \ldots & r_{l,d} \end{pmatrix} \bigodot \begin{pmatrix} c_{i1} \\ c_{i2} \\ \ldots \\ c_{id} \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \ldots \\ z_l \end{pmatrix}, \tag{1}$$

where $\bigodot$ is equivalent to matrix multiplication in binary operations. Specifically, for each cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$, we calculate:

$$z_k = \prod_{\oplus}{}_{j=1}^{d} \#(r_{k,j} \wedge c_{ij}),\ where\ 1 \leq k \leq l.$$

$\mathbf{z} = \langle z_1, z_2, \ldots, z_l\rangle$ will be a string of 0 and 1, which is the address of the entry where cell $\mathbf{c_i}$ is located in the hash table. By the definition of operator $\bigoplus$ and $\#$, we have

$$\prod_{\oplus}{}_{j=1}^{d} \#(r_{k,j} \wedge c_{ij}) = \#(\prod_{\oplus}{}_{j=1}^{d} (r_{k,j} \wedge c_{ij})).$$

We can take advantage of this commutative property of the operators to calculate the hash function efficiently. The following theorem can give us more insight into the time complexity of applying the hash function.

Theorem 1 *The calculation of hashed index using Equation 1 takes $O(ld)$, where $l = \lceil \log n \rceil$ is the minimum number of bits required to represent the address of each bucket and d is the number of dimensions.*

Proof: Let ρ be the length of any of the binary vectors $r_{k,j}$ and c_{ij}. We assume $\rho = O(\iota)$, where ι is the length of memory word and it is a small constant. The operation $r_{k,j} \wedge c_{ij}$ takes $O(\rho) = O(1)$ time. Let $\xi = \prod_{\oplus}{}_{j=1}^{d} (r_{k,j} \wedge c_{ij})$. It takes $O(d)$ to compute ξ. Also, $\#(\xi)$ can be computed in constant time. Thus, it takes $O(l(d+\rho))$ time to compute $\mathbf{z} = \langle z_1, z_2, ..., z_l\rangle$, where $\mathbf{z}$ is the address of $\mathbf{c_i}$ and ρ is a constant. Therefore, $O(l(d+\rho)) = O(ld)$. □

Hash collisons. The problem with the hash data structure is that two or more cells may be hashed into the same bucket, which is called **collision**. We resolve collision by extended queuing with each bucket.

Properties of hash function used. Given a hash table T with n buckets that stores N' elements, we define the *load factor* α for T as N'/n. That is, α is the average number of cells stored in each of the extended queues.

In a hash table structure, if any element is equally likely to be hashed into any of the m buckets independent of where any other element has been hashed to, we call this *simple uniform hashing.* It can be proved that the hash function given in Equation 1 maps a cell in any hash bucket with equal probability is a *simple uniform hash function.*

Below we discuss two important properties of the hash data structure which will be useful in the complexity analysis in Section 4.

Property 1 *In a simple hash table in which collisions are resolved by queuing, an unsuccessful search takes* $\Theta(t(h(c_i)) + d\alpha)$*, where* $t(h(c_i))$ *is the time that is taken to calculate* $h(c_i)$*.*

Proof: In [2] it has been shown that the time complexity of calculating the hash function and performing search to find an element is $\Theta(1+\alpha)$. In our hash function, it takes $t(h(c_i))$ to calculate hash function and it takes $O(d)$ to check a single item in the hash table. Therefore, an unsuccessful search takes $\Theta(t(h(c_i)) + d\alpha)$. □

Property 2 *In a simple hash table in which collisions are resolved by queuing, a successful search takes* $\Theta(t(h(c_i)) + d\alpha)$*.*

Proof: Similar to the proof of Property 1.

3.3 Neighborhood Determination in Hash Structure

In multidimensional array implementation, if the address of a cell is known, then finding the addresses of the neighboring cells is straightforward. Since simple uniform hashing permits any element to be mapped into any of the hash table buckets, it introduces the problem of determining the neighbor of a cell. The neighborhood information is no longer obvious. Given the location of a cell $\mathbf{c_i}$ its neighbors' bucket location can be found by computing the hash function on the indices of the neighbors. This takes $O(k \times l \times d) = O(l \times d)$, where k is the number of neighbors (which is a small constant).

Since both applying wavelet transform and finding connected components require the neighborhood information very frequently, an efficient method to determine the neighbors will have positive effect on the overall running time. We describe a simple technique to rediscover the neighborhood information efficiently.

Theorem 2 *Suppose* $\mathbf{c_j} = \langle c_{j1}, c_{j2}, \ldots, c_{jk}, \ldots, c_{jd} \rangle$ *is a cell and* $\mathbf{c'_j} = \langle c_{j1}, c_{k2}, \ldots, c'_{jk}, \ldots, c_{jd} \rangle$ *is an* ϵ*-neighbor of* $\mathbf{c_j}$*, where* $|c_{jk} - c'_{jk}| \leq \epsilon$*. The hashed index value of* $\mathbf{c'_j}$ *can be computed using the following formula:*

$$h(c'_j) = h(c_j) \bigoplus \begin{pmatrix} \#(c_{jk} \wedge r_{1,k}) \\ \#(c_{jk} \wedge r_{2,k}) \\ \ldots \\ \#(c_{jk} \wedge r_{d,k}) \end{pmatrix} \bigoplus \begin{pmatrix} \#(c'_{jk} \wedge r_{1,k}) \\ \#(c'_{jk} \wedge r_{2,k}) \\ \ldots \\ \#(c'_{jk} \wedge r_{d,k}) \end{pmatrix}. \tag{2}$$

Proof: We use $\zeta_{\mu,\lambda}$ to represent $\#(r_{\mu,\lambda} \wedge c_{j\,\lambda})$ and $\zeta'_{\mu,\lambda}$ to represent $\#(r_{\mu,\lambda} \wedge c'_{j\,\lambda})$.

$$
\begin{aligned}
h(c'_j) &= \begin{pmatrix} \#(r_{1,1} \wedge c_{j\,1}) \bigoplus \#(r_{1,2} \wedge c_{j\,2}) \bigoplus \ldots \bigoplus \#(r_{1,k} \wedge c_{j'\,k}) \bigoplus \ldots \bigoplus \#(r_{1,d} \wedge c_{j\,d}) \\ \#(r_{2,1} \wedge c_{j\,1}) \bigoplus \#(r_{2,2} \wedge c_{j\,2}) \bigoplus \ldots \bigoplus \#(r_{2,k} \wedge c_{j'\,k}) \bigoplus \ldots \bigoplus \#(r_{1,d} \wedge c_{j\,d}) \\ \ldots \\ \#(r_{l,1} \wedge c_{j\,1}) \bigoplus \#(r_{l,2} \wedge c_{j\,2}) \bigoplus \ldots \bigoplus \#(r_{l,k} \wedge c_{j'\,k}) \bigoplus \ldots \bigoplus \#(r_{l,d} \wedge c_{j\,d}) \end{pmatrix} \\
&= \begin{pmatrix} \zeta_{1,1} \bigoplus \zeta_{1,2} \bigoplus \ldots \bigoplus \zeta'_{1,k} \bigoplus \ldots \bigoplus \zeta_{1,d} \\ \zeta_{2,1} \bigoplus \zeta_{2,2} \bigoplus \ldots \bigoplus \zeta'_{2,k} \bigoplus \ldots \bigoplus \zeta_{2,d} \\ \ldots \\ \zeta_{l,1} \bigoplus \zeta_{l,2} \bigoplus \ldots \bigoplus \zeta'_{l,k} \bigoplus \ldots \bigoplus \zeta_{l,d} \end{pmatrix} \\
&= \begin{pmatrix} \zeta_{1,1} \bigoplus \zeta_{1,2} \bigoplus \ldots \bigoplus \zeta_{1,k} \bigoplus \zeta_{1,k} \bigoplus \ldots \bigoplus \zeta_{1,d} \bigoplus \zeta'_{1,k} \\ \zeta_{2,1} \bigoplus \zeta_{2,2} \bigoplus \ldots \bigoplus \zeta_{2,k} \bigoplus \zeta_{2,k} \bigoplus \ldots \bigoplus \zeta_{2,d} \bigoplus \zeta'_{2,k} \\ \ldots \\ \zeta_{l,1} \bigoplus \zeta_{l,2} \bigoplus \ldots \bigoplus \zeta_{l,k} \bigoplus \zeta_{l,k} \bigoplus \ldots \bigoplus \zeta_{l,d} \bigoplus \zeta'_{l,k} \end{pmatrix} \\
&= \begin{pmatrix} \zeta_{1,1} \bigoplus \zeta_{1,2} \bigoplus \ldots \bigoplus \zeta_{1,k} \bigoplus \ldots \bigoplus \zeta_{1,d} \\ \zeta_{2,1} \bigoplus \zeta_{2,2} \bigoplus \ldots \bigoplus \zeta_{2,k} \bigoplus \ldots \bigoplus \zeta_{2,d} \\ \ldots \\ \zeta_{l,1} \bigoplus \zeta_{l,2} \bigoplus \ldots \bigoplus \zeta_{l,k} \bigoplus \ldots \bigoplus \zeta_{l,d} \end{pmatrix} \bigoplus \begin{pmatrix} \zeta_{1,k} \\ \zeta_{2,k} \\ \ldots \\ \zeta_{l,k} \end{pmatrix} \bigoplus \begin{pmatrix} \zeta'_{1,k} \\ \zeta'_{2,k} \\ \ldots \\ \zeta'_{l,k} \end{pmatrix} \\
&= h(c_j) \bigoplus \begin{pmatrix} \#(c_{j\,k} \wedge r_{1,k}) \\ \#(c_{j\,k} \wedge r_{2,k}) \\ \ldots \\ \#(c_{j\,k} \wedge r_{d,k}) \end{pmatrix} \bigoplus \begin{pmatrix} \#(c_{j'\,k} \wedge r_{1,k}) \\ \#(c_{j'\,k} \wedge r_{2,k}) \\ \ldots \\ \#(c_{j'\,k} \wedge r_{d,k}) \end{pmatrix}. \qquad \square
\end{aligned}
$$

Theorem 3 *The calculation of hashed index using Equation 2 takes $O(\log n(\rho))$, where ρ is the length of any binary vectors of $r_{k,j}$ and $c_{i\,j}$ and n is the number of buckets in the hash table. Since ρ is a constant, the time complexity can be written as $O(\log n)$.*

Proof: Again, it takes $O(\rho) = O(1)$ to compute $\#(r_{j,i} \wedge c_{j\,i})$. $h(c_j)$ is already known, so it takes $O(\log n)$ to compute $h(c'{}_j)$ in Equation 2. $\square$

From Theorems 1 and 3 we see that the time complexity of calculating neighbors' indices is reduced by a factor of d, which is a significant improvement in running time for high dimensional cases.

3.4 Applying Wavelet Transform in High Dimension

Wavelet transform is a type of signal representation that can give the frequency content of the signal at a particular instant of time by filtering. A one-dimensional signal s can be filtered by convolving the filter coefficients c_k with the signal values. We have the following formula:

$$\hat{s}_i = \sum_{k=0}^{M-1} c_k s_{i+k-\frac{M}{2}}, \tag{3}$$

where M is the number of coefficients in the filter and $\hat{s}$ is the result of convolution.

Applying wavelet transform on a signal decomposes it into different frequency sub-bands [6]. We now briefly review wavelet-based multi-resolution decomposition. More details can be found in Mallat's paper [7]. We can compute a coarser approximation of the one-dimensional input signal S_0 by convolving it with the low pass filter $\tilde{H}$ and down sampling the signal by two [7]. All the approximations S_j, $1 < j < J$ (J is the maximum possible scale), can thus be computed from S_0 by repeating this process.

We can extract the difference of information between the approximation of signal at scale $j-1$ and j. D_j denotes this difference of information and is called *detail signal* at the scale j. We can compute the detail signal D_j by convolving S_{j-1} with the high pass filter $\tilde{G}$ and returning every other sample of output. The wavelet representation of a discrete signal S_0 can therefore be computed by successively decomposing S_j into S_{j+1} and D_{j+1} for $0 \leq j < J$. This representation provides information about signal approximation and detail signals at different scales.

The above wavelet model can be generalized for d-dimensional feature space, where one-dimensional wavelet transform will be applied d times. That is, along each of the d dimensions of the feature space, we convolve with the low pass and high pass filters and down sample by two. It results in one approximation, or *average subband (feature space)*, and $2^d - 1$ *detail subbands (feature spaces)*. At different scales, approximations (average subbands) have information about content of clusters, and detail subbands have information about boundaries of clusters. In the experiments, we apply Haar, Daubechies, Cohen-Daubechies-Feauveau ((2,2) and (4,2)) wavelet transforms [13, 11, 12] and compare their effectiveness on clustering.

3.5 Calculating Wavelet Transform on Hashed Feature Space

Wavelet transform is applied on hashed representation of the quantized space to generate a new hash table consisting of only the significant cells. By scanning through the hash table and convolving the filter given in Equation 3 with each cell and its neighbors, we generate new cells in the transformed space. For any nonempty cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$, the cells which will contribute to its value in the transformed space along dimension j are, $\mathbf{c_k} = \langle c_{i1}, c_{i2}, \ldots, c_{ij} + k, \ldots, c_{id} \rangle$, where $\frac{M}{2} \leq k \leq \frac{M}{2}$ and M is the width of the wavelet filter. All the cells stored in the hash table will get new values after wavelet transform is applied. Also, because of the convolution operation in wavelet transform, some of the previously empty cells will become nonempty by receiving contributions from their neighboring cells. We call each potential nonempty cell a *receiver* and each old nonempty cell a *contributor*. In traditional implimentation of wavelet transform, each receiver knows which cells to ask for contributions. Thus, the algorithm scans through all the potential receivers. In a multidimensional array implementation, every cell is considered as a potential receiver. So the algorithm has to scan through the entire space of cells which we should try to avoid in high dimension case because of the exponential growth in the number of cells. But, in the case of hashed implementation we only have information about the cells which are nonempty. So there are many potential receivers about whose hashed location we have no knowledge. Therefore, it is not possible to use traditional algorithms directly on the hashed quantized feature space.

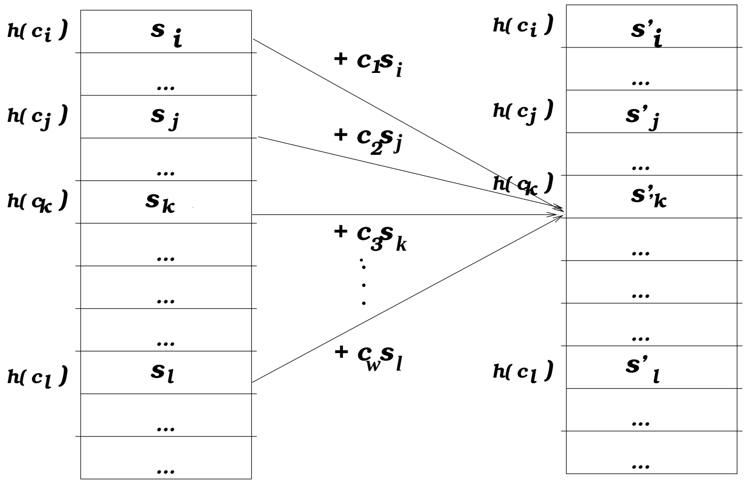

Figure 1: Traditional approach of calculating wavelet transform.

In our approach, each *contributor knows* its *receivers* and the addresses of the *receivers* can be calculated by using the hash function in $O(l)$ (from Property 3). So, instead of receivers asking for values, contributors distribute values to receivers. Thus, it is sufficient to just scan through the contributors which are already saved in the hash table.

With slight modifications, we can rewrite Formula 3 as:

$$\widehat{s_{i+j}} = \sum_{k=0}^{\frac{M}{2}-j-1} c_k s_{i+j-\frac{M}{2}+k} + \mathbf{c}_{\frac{\mathbf{M}}{\mathbf{2}}-\mathbf{j}}\mathbf{s}_{\mathbf{i}} + \sum_{k=\frac{M}{2}-j+1}^{M-1} c_k s_{i+j-\frac{M}{2}+k}, \tag{4}$$

where $-M/2 \leq j < M/2$.

Using this formula while scanning the hash table, each old nonempty cell or contributor is multiplied by a coefficient and the result is accumulated into its *receiver* cells which are hashed into a new table.

Due to the generation of new nonempty cells, the number of cells in the new hash table will be increased after wavelet transform is applied. In many cases, a large number of new nonempty cells tend to have very small count values. Many of these low count values are expected to be caused by the outliers rather than the actual clusters. Also, the actual cluster shapes are distorted on the

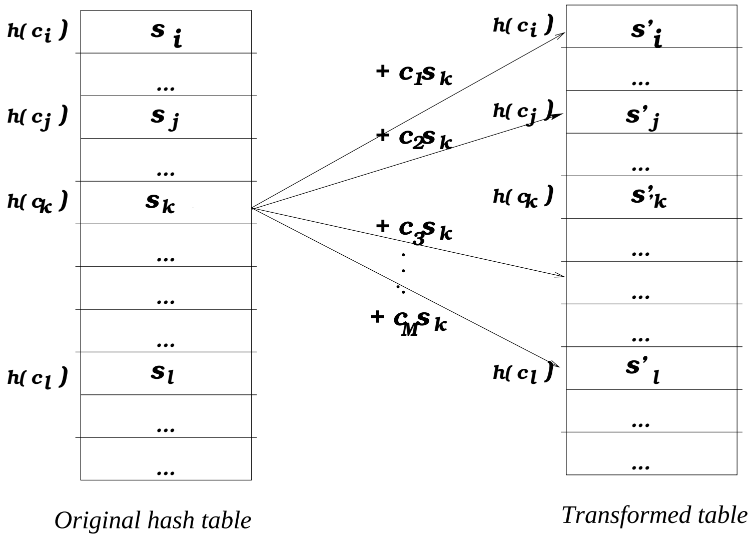

Figure 2: Accumulative approach of calculating wavelet transform.

surfaces because of the directionality property of convolution operation used in wavelet transform. Removing low count value cells by applying a threshold on the count values will effectively remove majority of the outliers and help preserving the original shape of the clusters. In addition, reduction in number of cells in the hash table is expected to improve the time complexity of the algorithm. In Section 2, we defined the significant cell as a cell which has count values greater than a particular threshold τ. In the new hash table constructed after applying wavelet transform on the original hash table, only the significant cells are stored. The threshold τ plays an important role in the quality of clustering and outlier removing.

3.6 Thresholding

In this section we discuss issues in specifying the appropriate threshold τ for finding the significant cells in the transformed space. Let $g_i, 1 \leq i \leq d$, denote the factor of increase in number of nonempty cells when one-dimensional wavelet transform is applied to the quantized space along dimension i. Here, g_i depends on the wavelet filter width M and the characteristic of data distribution along dimension i. Without data dropping, the final number of nonempty cells will be $(\prod_{i=1}^{d} g_i)N'$, where N' is the original number of nonempty cells in the quantized space. We perform thresholding on the count values of the nonempty cells in the transformed space to bring down the number of nonempty cells and keep only the significant cells. Mathematically, after applying wavelet transform along one dimension and applying threshold, the total number of cells become $\beta N'$, where β is called the *control factor* and can be set appropriately. So total number of cells after applying wavelet transform on all the dimensions is $\beta^d N'$. In [8], Matias et. al. have shown how to find the optimal threshold factor τ. In HiperWave, we use a histogram-based method to find an approximation of the optimal threshold. A histogram of the count values is constructed as shown in Figure 3. The threshold is set so that the total number of cells on the right of the threshold is $\beta N'$. For a β

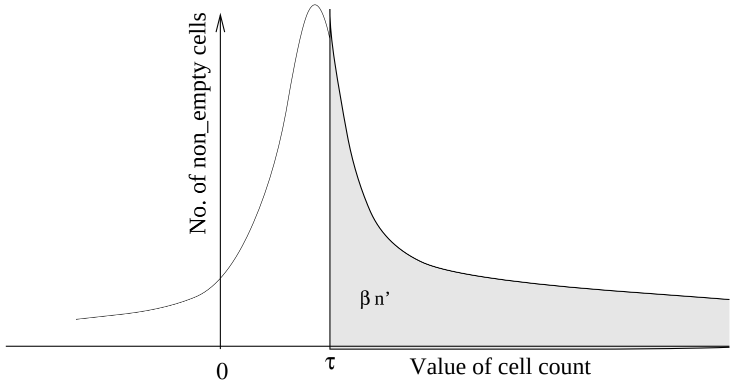

Figure 3: The histogram-based threshold calculation.

value of less than 1, we actually gain in term of time complexity as the dimensionality of the data increases. Obviously, such a β value will have negative effect on the quality of the clusters, i.e., many of the real clusters will be deleted as noise. A large β value will keep all the real clusters, but will be ineffective in removing outliers. Also, the surface distortion of the high dimensional clusters will not be compensated. As β increases, the complexity of the algorithm increases by β^d. From our experience, a β value close to 1 works well in most of the cases.

3.7 Finding Connected Components in Hash Table

The hash table is essentially a graph $G = (V, E)$, where $V = \{\mathbf{c_i}$ — c_i is a significant cell in transformed space$\}$ and $E = \{(c_1, c_2) | D(c_1, c_2) \leq \varepsilon = 1\}$. Here, we define distance D as *City-block distance* or $\|l_1\|$ metric:

$$D_{\|l_1\|}(\mathbf{c_1}, \mathbf{c_2}) = \sum_{i=1}^{d} \mid c_{1i} - c_{2i} \mid .$$

There is an edge between two **cells** if and only if their indices differ on only one dimension. So every significant cell has at most $2d$ neighbors. Given the index of a cell, the bucket number of the neighboring cells can be computed by the hash function in Equation 2. The *clusters* are the *connected components* of graph G, which can be found by a depth-first-search algorithm. We start from the first bucket in the hash table, assign it the first cluster number and search the cells it is connected to. Continue to scan the *hash table* until all the cells are visited.

3.8 Labeling Cells and Making Look Up Table

Applying wavelet transform on the cells in $\{\mathbf{c_j} : 1 \leq j \leq \mathcal{J}\}$ results in a new feature space and hence new cells $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$. Given the set of cells $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$, HiperWave detects the connected components in the transformed feature space. Each connected component is a set of cells in $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$ and is considered as a cluster. Each cluster w, $w \in \mathcal{C}_r$, will have a cluster number w_n. In this step we label the cells in the original quantized feature space that are included in a cluster, with its cluster number. That is,

$$\forall w \ \ \forall \mathbf{t_k}, \ \mathbf{t_k} \in w \Longrightarrow l_{t_k} = w_n, \quad w \in \mathcal{C}_r,$$

where l_{t_k} is the label of the cell t_k. The clusters that are found are in the transformed space. Thus, they cannot be directly used to define the clusters in the original space. We make a lookup table LT which is also a hash table to map the units in the transformed feature space to the units in the quantized space. Each entry in the table specifies the relationship between a cell in the transformed space and the corresponding cell(s) in the quantized space. So the label of each cell in the original space can be easily determined. Finally the label of each unit in the quantized space is assigned to all the objects contained in that cell, and thus the clusters are determined. Formally,

$$\forall w \ \ \forall \mathbf{c_j}, \ \ \forall \mathbf{o_i} \in \mathbf{c_j}, \qquad l_{o_i} = w_n, \quad w \in \mathcal{C}_r, \ \ 1 \leq i \leq N,$$

where l_{o_i} is the cluster label of object o_i.

4 Complexity Analysis

In this section, we discuss the time and space complexity of HiperWave. Let N be the number of d-dimensional objects in the database, usually N is a very large number. The following notation is needed for the ensuing discussion:

m - Number of quantization levels.
M - Width of the convolution filter used in wavelet transform.
n - Number of buckets in the hash table.
N' - Number of nonempty cells after quantization.
α - Load factor as defined in Section 3.2.
g - Growth rate of number of nonempty cells after applying wavelet transform (constant along each dimension) in one direction.
β - The control factor (as defined in Section 3.6).

4.1 Time Complexity

The quantization step consists of scanning through the N data vectors, each of which has d components. For each data vector we determine the cell containing the vector and calculate the position of this cell in the hash table. The calculation of hashed index for each data vector takes $O(d\ logn)$, following Theorem 1. From Property 1 the time taken to search for a space for each data vector is $O(d \log n + d\alpha)$. Thus, the total time taken in quantization and creating the hash table is

$$O(N(d\ logn + d\alpha)) = O(Nd\ logn). \tag{5}$$

Here we assume that $N' = O(n)$ so that the load factor α is a constant.

In calculating wavelet transform we use the modified hash function as given in Equation 2. Calculation of wavelet transform in each dimension consists of scanning through all the elements stored in the hash table, which is same as $n\alpha$, and for each element we calculate the modified hash function. Using Property 2 and Theorem 3 the time taken for each calculation of the hash function and searching in the new hash table takes $O(d\ logn + d\alpha) = O(d\ logn)$ time. Also, applying wavelet transform on $n\alpha$ cells for the first dimension takes $O(Mn\alpha\ logn)$ time, since M and α are both constants. The thresholding requires another scan through all the elements in the newly created hash table which has $gn\alpha$ entries. Thus, it takes $O(gn\alpha)$ time. Thresholding brings down the number of cells to $\beta n\alpha$. Therefore after applying wavelet transform in all d dimensions the total number of significant cells become $\beta^d n\alpha$. The time taken to calculate wavelet transform and thresholding on $\beta^d n\alpha$ cells is $O(M\beta^d n\alpha(logn + d\beta^d\alpha))$, which can be simplified to $O(\beta^d n(logn + d\beta^d))$ since α and M are constants. Total time taken in applying wavelet transform in all d-dimensions is just summation of the time taken in applying wavelet transform in each dimension. When $\beta \geq 1$, the maximum number of cells on which wavelet transform is applied at any step is $\beta^d n\alpha$ (in the last step). We can approximate the running time of applying wavelet transform on d-dimension by the time taken to apply wavelet transform independently d times on $\beta^d n\alpha$ cells, which is really a very loose upper bound for the actual time complexity. In case of $\beta \leq 1$ the number of cells decrease continually as we apply wavelet transform on each dimension. So a loose upper bound in this case can be the time complexity of applying wavelet transform d times on original $n\alpha$ cells. Considering only the case of $\beta \geq 1$ the worst case time complexity of calculating wavelet transform and thresholding in d-dimensions can be given as,

$$O(d\beta^d n(logn + d\beta^d)). \tag{6}$$

As discussed in Section 3.7, the connected component analysis is same as depth-first search in graph with $V = \beta^d n\alpha$ nodes and $E = d\beta^d n\alpha$ edges. But our implementation of hash table makes it slightly different from an ordinary graph representation. It requires $O(logn + d\alpha)$ time to search each neighbor instead of O(1) as found in usual graph algorithms. Therefore, time taken for the depth first seach is, $O(d\beta^d n(logn + d\alpha))$ which is asymptotically equal to :

$$O(\beta^d nd(logn + d)). \tag{7}$$

The labeling and making look up table takes only a single scan through all the cells stored in the hash table, which is same as

$$O(\beta^d n\alpha) = O(\beta^d n). \tag{8}$$

From Equations 6, 7 and 8, we observe that the running time increases exponentially with dimension d on β. We choose $\beta = 1$ to avoid this problem. Substituting this value of β in Equations 6, 7 and 8 and comparing the result with Equation 5, we can clearly see that overall running time is $O(Nd\ logn)$ for the case of $\beta < 1$. For the case of $\beta \geq 1$ the time complexity is bigger of $O(\beta^d n(logn + \beta^d d))$ and $O(Nd\ logn)$. Choosing $\beta = 1$ we find the overall time complexity is $O(dn\ logn + d^2 n + Ndlogn)$. In the average case we expect $d^2 n\ logn << Ndlogn$, because we expect clusters to exist only in very small fraction of the total number of cells in the quantized space. So the expected running time is, $O(Ndlogn)$.

So by introducing the hash data structure to represent the dataset, we can cluster high dimensional data within a $logn$ factor of the input data size. In the worst case, every data object maps to a distinct cell during quantization, i.e., $N' = N$. Hence the worst case complexity of HiperWave will be $O(dN\ logN + d^2 N)$. But we must remember that this is an extreme case and in such a case there is actually no cluster in the original dataset, the data objects are uniformly spread over the entire feature space.

4.2 Space Complexity

The size of each cell saved in the hash table is $O(d)$, since the d-dimensional address of the cell (in quantized space) must be saved in each hash table location to uniquely identify the cell. The maximum space requirement of HiperWave is $O(\beta^d n\alpha d)$, which is same as maximum number of cells we need to keep in memory during the calculation of wavelet transform. This is again exponential on the control factor β. Choosing $\beta = 1$, we limit the space complexity to $O(dn\alpha) = O(dN')$ which is linear with respect to the number of nonempty cells after quantization. This is much less than total number of all possible cells m^d in the high dimensional case.

5 Experiments

5.1 Synthetic Data Generation

We designed our own synthetic dataset generator for performing experiments. The data generator allows control over the structure and size of datasets. The user can specify number of spatial data points, number of clusters, number of dimensions, number of points in each cluster, range of values for each attribute and the underlying probability distribution of the data points in each cluster. The generator also allows the user to specify number of noise points. The noise points are spread following an uniform random distribution across the space.

The clusters are either hyper-rectangles (following a uniform random distribution) or spheres (following d-dimensional independent normal random distribution). The density of clusters are much higher than the average density of the space around the cluster. The generation of hyper-rectangle clusters follow closely the method described in [15]. For spherically shaped clusters the user specifies the central point (or mean of the distribution) in each cluster. Around the cluster center, points are spread following a d-dimensional independent normal distribution. For simplicity we assumed that each dimension is independent. The user has the flexibility to choose the variance in each dimension. Due to the properties of normal distribution, the maximum distance between a point in the cluster and the center is unbounded. So a data point that belongs to cluster A may be closer to some other cluster B, we refer such points as *outliers* too. By fixing proper threshold in the wavelet transformed space all such points are effectively removed.

We generated 14 different synthetic data sets to evaluate different aspects of our approach. The datasets have sizes from 100,000 to 1,000,000 objects with 5 to 20 dimensions (features). We added noise objects to the datasets that make about 10% of the dataset objects. Table 1 summarizes the information about the datasets used in our experiments. All the experiments were carried out on a SUN SPARC workstation using 168 MHz UltraSparc CPU with SunOS operating system and 1024 MB memory.

Dataset	Number of Dimensions	Number of Data Points	Number of Outliers	Number of Clusters	Comments
1	12	100,000	10,000	9	9 hypercube clusters
2	12	200,000	20,000	9	9 hypercube clusters
3	12	300,000	30,000	9	9 hypercube clusters
4	12	400,000	40,000	9	9 hypercube clusters
5	12	500,000	50,000	9	9 hypercube clusters
6	12	600,000	60,000	9	9 hypercube clusters
7	12	700,000	70,000	9	9 hypercube clusters
8	12	800,000	80,000	9	9 hypercube clusters
9	12	900,000	90,000	9	9 hypercube clusters
10	12	1000,000	100,000	9	9 hypercube clusters
11	5	100,000	10,000	9	5 hypercube and 4 spherical clusters
12	10	100,000	10,000	9	5 hypercube and 4 spherical clusters
13	15	100,000	10,000	9	5 hypercube and 4 spherical clusters
14	20	100,000	10,000	9	5 hypercube and 4 spherical clusters

Table 1: Test datasets used in the experiments.

5.2 Quality of Clustering

In the existing clustering papers, the efficiency and required processing time of the clustering methods are accurately reported. However, not much effort has been put to measure the quality of the clustering. Usually the quality of clustering is presented just by visualizing the clusters in the feature space and no quantitative measurement of quality is presented. Visualization methods can be useful for low dimensional feature spaces (2 or 3 dimensions). But for higher number of dimensions they may not be applicable. Moreover, they cannot directly be used to accurately compare different clustering methods. Number of detected clusters has also been used to show the quality of clustering. But this number alone cannot be a good indication for quality. We should consider how precisely the detected clusters match the real clusters in the feature space. We measured the effectiveness of HiperWave in terms of the number of correct clusters found and *precision of clustering*. We use the labels of objects in the dataset to define a metric for the quality measurement. During data generation, each object o_i will have a desired label dl_{o_i} indicating the cluster it belongs to. For synthetic datasets, this can be done automatically while the data is generated. In the experiments for real datasets, domain knowledge is required to assign the desired label to the dataset objects. After clustering, HiperWave assigns label l_{o_i} to the objects o_i based on the clusters that it detects. Precision of clustering P is defined as

$$P = \frac{Number\ of\ correctly\ labeled\ objects}{Total\ number\ of\ objects\ in\ dataset}.$$

The object o_i is correctly labeled if $dl_{o_i} = l_{o_i}$. A higher value for P indicates that the clustering method can correctly label more number of objects in the clusters, and thus has better quality of clustering. All the outliers should have the same desired label. Thus, correctly identifying outliers increases the value of P which shows the improvement in the quality of clustering.

Dimensionality of dataset. We tried HiperWave for datasets with different number of dimensions. Table 2 shows the precision of clustering for datasets of size 100,000 objects. It shows that for dimensions below 20, HiperWave can identify almost all the clusters with high precision. As the number of dimensions increases, it would be more difficult to find the clusters correctly and thus precision decreases.

Dimension of Data	Number of Clusters	Clusters Found	True Clusters Found	Precision
5	9	8	8	0.89
10	9	23	9	0.90
12	9	13	9	0.76
15	9	8	8	0.75
20	9	11	6	0.57

Table 2: Precision of clustering for different number of dimensions.

Wavelet transform. We applied different types of wavelet transform on the feature spaces. Table 3 summarizes the performance of HiperWave for four transforms: Haar, Daubechies (with 6 coefficients), Cohen-Daubechies-Feauveau ((2,2) and (4,2)) [13, 11, 12]. The experiments were performed on 12-dimensional datasets with 70,000 objects. As we expected, because of (Mexican hat) shape of Cohen-Daubechies-Feauveau ((2,2) and (4,2)) filters, we can achieve better performance than that of Haar and Daubechies [10]. In our experiments, Cohen-Daubechies-Feauveau (2,2) has the best performance among other transforms. The required time to apply wavelet transform is a factor of its filter length. The longer the filter is, the more time it requires to do the computations.

Wavelet Transform	Time (sec.)	Number of Clusters	Clusters Found	True Clusters Found	Precision
Haar	1,057	9	9	8	0.12
Daubechies	11,114	9	10	8	0.19
Cohen-Daubechies-Feauveau (2,2)	9,749	9	13	8	0.69
Cohen-Daubechies-Feauveau (4,2)	10,192	9	13	8	0.55

Table 3: Execution time (in seconds) and precision of clustering for different wavelet transforms.

Quantization. In general, when the quantization value m is too low (very coarse quantization), more objects will be assigned to the same cell, and there is higher probability for the objects from different clusters to belong to the same cell. We can call this case *under-quantization* problem. This results in merging of the two clusters and mislabeling their objects, thus the precision of clustering decreases. In contrast, if the quantization value m is too high (very fine quantization), each object will be in a separate cell which might be far from the other cells. We call this *over-quantization* problem. Over-quantization can result in many unnecessary small clusters (that might be later removed as outliers) and does not find the real clusters, thus it will also decrease the precision of clustering. Precision of clustering for different number of dimension (from 5 to 20) and different quantizations (from 32 to 256) for datasets with 100,000 objects is presented in Table 4.

Number of Dimensions	m				
	32	64	96	128	256
5	0.11	0.70	0.70	0.70	0.88
10	0.40	0.49	0.80	0.69	0.60
15	0.63	0.69	0.40	0.38	0.42
20	0.63	0.54	0.21	0.21	0.10

Table 4: Precision of clustering for different number of dimensions and quantizations m.

Figure 4 shows the precision for different quantizations. If we have the appropriate quantization, we can have the precision which is higher than both cases where m is very low or very high. Table 4 and Figure 4 clearly show this behavior when the number of dimensions is 10. When the value of m is 32 we face the under-quantization problem and the precision is only 0.40. When m is 256, we have the over-quantization problem and the precision is 0.60. But when m is 96 we have the highest precision 0.80 which shows that $m = 96$ is an appropriate quantization. The appropriate quantization value depends on the dimensionality. As Table 4 and Figure 4 show, when number of dimensions is low, we should have a higher value for m to reach a good performance. But for higher number of dimensions, quantization value should be low. In other words, for higher number of dimensions, the over-quantization problem becomes worse and we should keep the quantization value m low to achieve better performance.

Outlier removal. One of the advantages of our method is its effectiveness in removing the

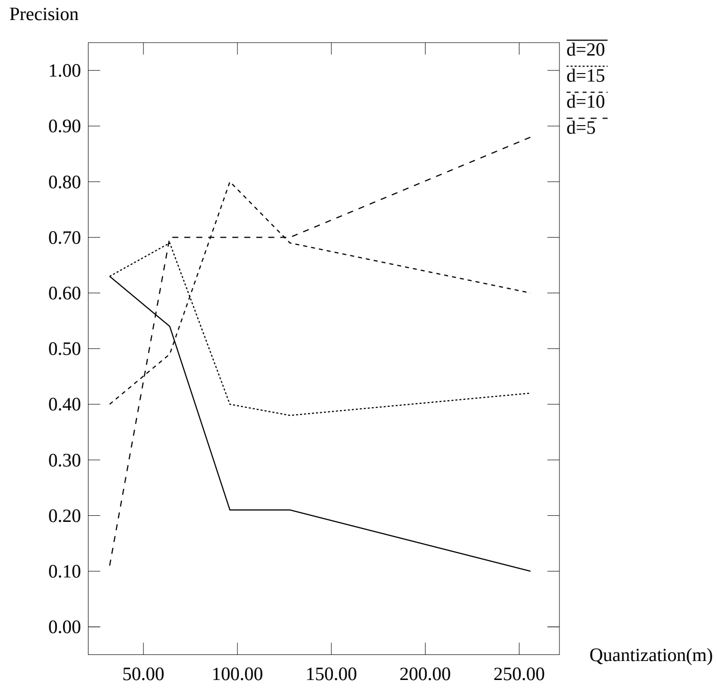

Figure 4: Precision w.r.t. quantization.

outliers. In our experiments, 10% of each dataset consisted of outliers which were distributed all over the feature space. HiperWave could successfully detect those outliers with 99% precision.

5.3 Efficiency

Dimensionality of datasets. Table 5 shows the required time to cluster datasets with 100,000 objects when applying Cohen-Daubechies-Feauveau (2,2) wavelet transform. Figure 5 shows the required time for different quantizations (m=32, 64, 96, 128, 256) when the dimensionality of the dataset changes. The value m indicates the number of intervals (cells) along each dimension d of the feature space. From Figure 5 and Table 5 we can observe that for each quantization value m, the required time increases linearly with respect to dimensionality.

Number of Dimensions	m = 32	64	96	128	256
5	118	132	216	256	340
10	1,042	1647	2,166	2,680	3,280
15	2,374	3958	7,332	7,156	8,080
20	3,742	6153	10,983	10,660	13,400

Table 5: Execution time (in seconds) for different number of dimensions and quantizations m.

Quantization. Table 5 also shows the required time when different quantization is used before clustering. There are m^d cells in the feature space. But as mentioned in Section 4.1, execution

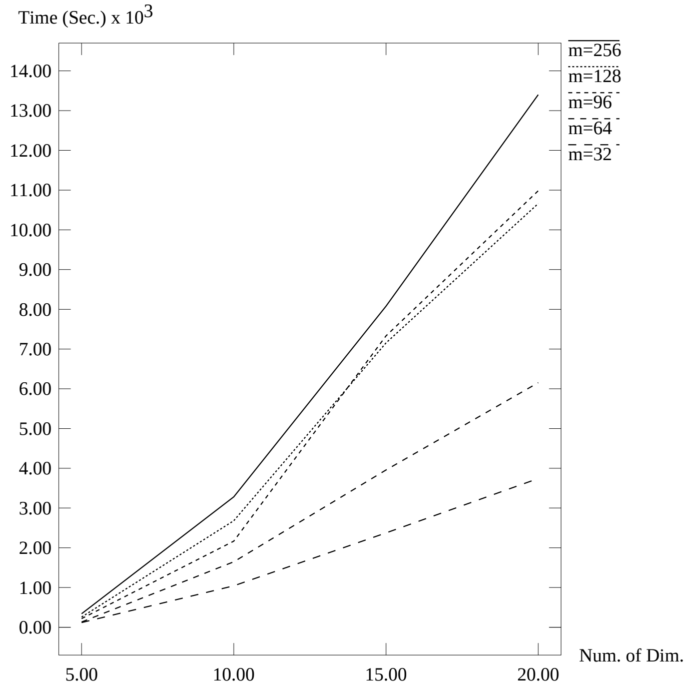

Figure 5: Execution time (in seconds) w.r.t. dimensionality.

time depends on the number of significant cells N'. Figure 6 shows the required time for different number of dimensions (d=5, 10, 15, 20) when quantization changes. When the quantization value m is low, the number of significant cells is low. By increasing the value of m, number of significant cells increases, and thus the required execution time increases too. As Table 5 and Figure 6 show, the increase in time is linear.

It should be noted that there is an upper bound for execution time. Because after some level of quantization, every object will be in one separate cell resulting in N significant cells. Further quantization of space will not change the number of significant cells, and the execution time of this case will be the upper bound.

Size of dataset. Figure 7 shows the required time for HiperWave to cluster datasets of sizes from 100,000 to 1,000,000 objects. These datasets are 12-dimensional and the value of m (quantization factor) is 64. As Figure 7 demonstrates, the required execution time increases almost linearly with respect to the size of datasets.

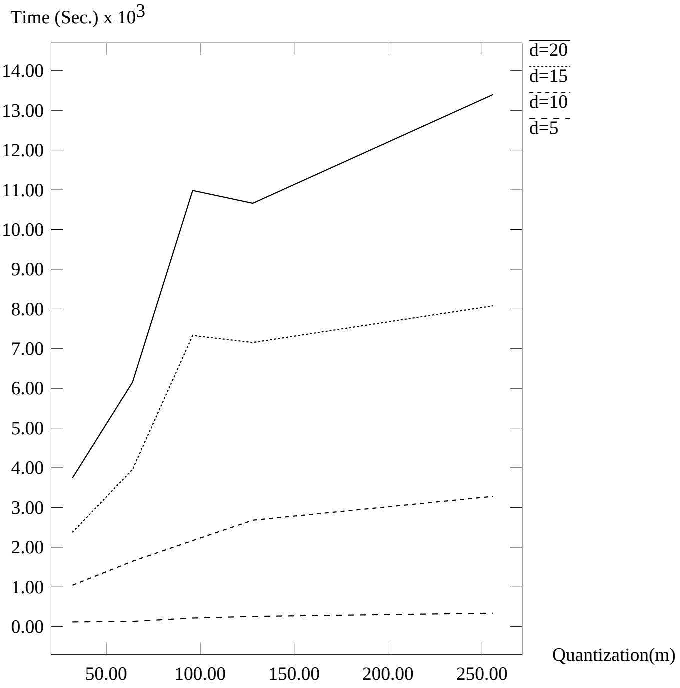

Figure 6: Execution time (in seconds) w.r.t. quantization.

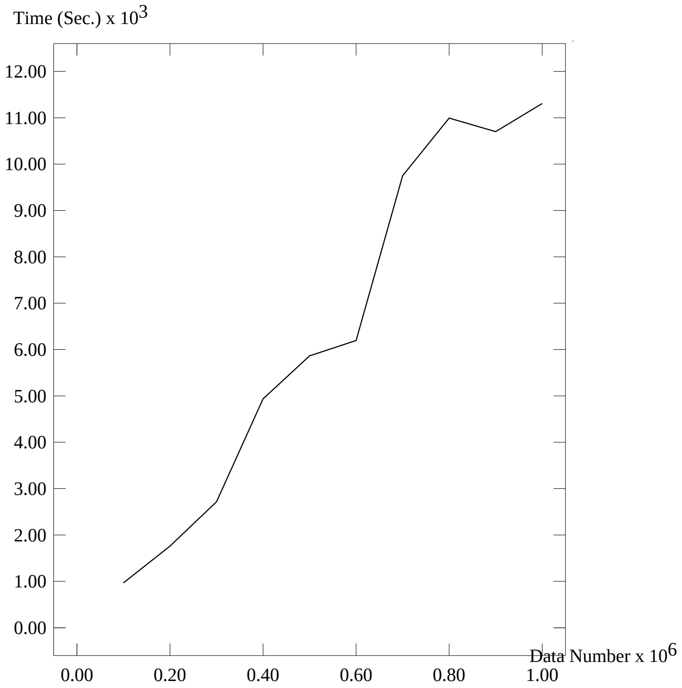

Figure 7: Execution time (in seconds) for datasets with different number of objects.

6 Conclusion

We have presented an approach, *HiperWave*, for efficiently clustering high dimensional data using wavelet transform. In this approach, we proposed a hash table data structure to represent the dataset. We then described a detailed technique to accumulatively apply wavelet transform on the hashed feature space. We demonstrated that this approach can intelligently utilize the available resources to efficiently perform clustering on high dimensional datasets. We also showed that the cost of clustering can be reduced dramatically yet maintaining all the advantages of wavelet-based clustering. Finally, we introduced a new quantitative metric to measure the quality of the resulting clusters. Our experimental results demonstrated that the proposed method can effectively and efficiently perform data clustering on a large dataset with high dimensions.

Acknowledgments

We would like to thank Dr. Jin-Yi Cai for his valuable input to the paper.

References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 94–105, Seattle, WA, 1998.

[2] Thomas H. Cormen, Charles E Leiserson, , and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[3] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of 2nd International Conference on KDD*, 1996.

[4] A. D. Gordon. *Classification Methods for the Exploratory Analysis of Multivariate Data.* Chapman and HAll, 1981.

[5] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD conference on Management of Data*, pages 73–84, Seattle, WA, 1998.

[6] S. Mallat. Multiresolution approximation and wavelet orthonormal bases of $L^2(R)$. *Transactions of American Mathematical Society*, 315:69–87, September 1989.

[7] S. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Trnasactions on Pattern Analysis and Machine Intelligence*, 11:674–693, July 1989.

[8] Yossi Matias, Jeffrey Vitter, and Min Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 448–459, Seattle, WA, 1998.

[9] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th VLDB Conference*, pages 144–155, Santiago, Chile, 1994.

[10] G. Sheikholeslami, S. Chatterjee, and A. Zhang. *WaveCluster*: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the 24th VLDB conference*, New York City, August 1998.

[11] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks.* Wellesley-Cambridge Press, 1996.

[12] Greet Uytterhoeven, Dirk Roose, and Adhemar Bultheel. Wavelet transforms using lifting scheme. Technical Report ITA-Wavelets Report WP 1.1, Katholieke Universiteit Leuven, Department of Computer Science, Belgium, April 1997.

[13] P. P. Vaidyanathan. *Multirate Systems And Filter Banks.* Prentice Hall Signal Processing Series. Prentice Hall, Englewood Cliffs, NJ,, 1993.

[14] Wei Wang, Jiong Yang, and Richard Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the 23rd VLDB Conference*, pages 186–195, Athens, Greece, 1997.

[15] Mohamed Zait and Hammou Messatfa. A comparative study of clustering methods. *Future Generation Computer Systems*, 13:149–159, November 1997.

[16] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, 1996.