RC23287 (W0408-022) August 6, 2004
Computer Science

IBM Research Report

An Efficient Subspace Sampling Framework for High Dimensional Data Reduction, Selectivity Estimation and Nearest Neighbor Search

Charu C. Aggarwal
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

An Efficient Subspace Sampling Framework for High Dimensional Data Reduction, Selectivity Estimation and Nearest Neighbor Search*

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

Abstract

Data reduction can improve the storage, transfer time, and processing requirements of very large data sets. One of the challenges of designing effective data reduction techniques is to be able to preserve the ability to use the reduced format directly for a wide range of database and data mining applications. In this paper, we propose the novel idea of hierarchical subspace sampling in order to create a reduced representation of the data. The method is naturally able to estimate the local implicit dimensionalities of each point very effectively, and thereby create a variable dimensionality reduced representation of the data. Such a technique is very adaptive about adjusting its representation depending upon the behavior of the immediate locality of a data point. An important property of the subspace sampling technique is that the overall efficiency of compression improves with increasing database size. Because of its sampling approach, the procedure is extremely fast and scales linearly both with data set size and dimensionality. We propose new and effective solutions to problems such as selectivity estimation and approximate nearest neighbor search. These are achieved by utilizing the locality specific subspace characteristics of the data which are revealed by the subspace sampling technique.

Keywords: High Dimensions, Dimensionality Reduction, Nearest Neighbor Search, Selectivity Estimation

*This paper is an extended version of [2].

1 Introduction

Recent advances in hardware technology have made it possible to collect very large amount of data. For example, even simple transactions of every day life such as using a credit card or the phone are logged in an automated way. Such data sets are often very high dimensional. Examples of such domains include supermarket data, multimedia data and telecommunication applications. This often results in massive data tables whose sizes are of the order of tera-bytes. In such cases, it is desirable to reduce the data in order to save on critical system resources such as storage space and transfer time of large files. In addition, many database applications can be implemented more efficiently on reduced representations of the data.

The dimensionality reduction problem has been well studied in the generic multi-dimensional context [29, 18, 12], or for particular data domains such as time series [9, 23, 24, 22, 32] and text [26]. In the generic case, a well known technique for dimensionality reduction is the method of Singular Value Decomposition [29, 18, 12] (SVD), which projects the data into a lower dimensional subspace. The idea is to transform the data into a new orthonormal coordinate system in which the second order correlations are eliminated. In typical applications, the resulting axis-system has the property that the variance of the data along many of the new dimensions is very small [18]. These dimensions can then be eliminated, a process resulting in a compact representation of the data with some loss of representational accuracy. Even though a variety of compression techniques [5, 16, 33] provide such guarantees, dimensionality reduction methods are more desirable because of their use of multidimensional representations for the compressed format. Furthermore, dimensionality reduction techniques are rarely used as stand alone methods for data compression. Rather, such representations allow the use of database applications such as indexing directly on the reduced representation without a first phase of reconstruction. This alternative often turns out to be significantly more efficient. On the other hand, the multidimensional representation is also a constraint which reduces the effectiveness of the reduction process. Furthermore, the high computational requirements of the dimensionality reduction method reduce the applicability of the approach for large and high dimensional databases.

Recent research has shown that even though the implicit dimensionality of a given data set may be quite high, particular subsets of it may show data dependencies which lead to much lower implicit dimensionality [3, 10]. An effective data reduction system would try to optimize the representation

of a record depending upon the distribution of the data in its locality. Clearly, it is a non-trivial task to find a representation in which each point adjusts its storage requirements naturally to the corresponding local implicit dimensionality. Since the issue of data reduction is most relevant in the context of large data sets, it is also necessary for the computational and representational requirements of such approaches to scale efficiently with increasing data size. Unfortunately, the techniques in [3, 10] are orders of magnitude slower than even the standard dimensionality reduction techniques, and are inflexible in determining the dimensionality of data representation. As a result, the applicability of these methods is restricted to specific applications such as indexing.

In recent years, the technique of random projection [1, 15, 26] has often been used as an efficient alternative for dimensionality reduction of high dimensional data sets. These techniques typically use spherically symmetric projections, in which arbitrary directions from the data space are sampled repeatedly in order to create a new axis system for data representation. While random projection is a much more efficient process than methods such as SVD, its average reduction quality is not quite as effective [8]. In this paper, we investigate the use of subspace sampling approaches in which the subspaces picked are determined by the (local) properties of the particular data set at hand. The use of a locality sensitive random sampling approach results in a system which is *both more effective and efficient* than SVD, while providing worst case bounds on the error loss of each record. The locality sensitive sampling method uses a hierarchical subspace sampling approach in which the storage requirement of each data point is influenced by the corresponding local implicit dimensionality. This variation from the global approach of standard dimensionality reduction methods has the interesting property that local implicit dimensionalities can be estimated more robustly for larger data sets. While the improvements of the sampling effectiveness with increasing number of database points are well known for many applications, this is not the case for global subspace sampling techniques [1, 17]. This is because previously proposed subspace sampling procedures [1, 8, 17, 15] sample dimensions rather than points, while each point must continue to be represented in the reduced format. For example, for the case of global random projection techniques [1, 17], the reduction factor may worsen [1] with increasing number of points. In this paper, we combine point sampling with space sampling in order to scale the effectiveness of the reduction technique with increasing database size. This is because the concept of data locality can be defined in a more refined way for larger data sets. Thus, one of the contributions of the paper is to leverage the power of sampling in such a way that the greater statistical information available in larger data sets is used more effectively. Some recent techniques such as LLE and ISOMAP [30, 31] use

locality concepts in the dimensionality reduction process. Although these techniques use locality properties in the reduction process, the aim of doing so is quite different. While the hierarchical subspace sampling technique attempts to find different (local) coordinate systems, the methods in [30, 31] try to use these in order to find a global projection system which will expose the desired (non-linear) properties of the data. A recent technique also uses data locality for reduction in the time series domain [24].

We will also show that the local characteristics of the data revealed by the hierarchical subspace sampling technique can be effectively leveraged for innovative solutions to problems such as selectivity estimation and nearest neighbor indexing. The selectivity estimation problem is motivated by the time-consuming nature of the query resolution problem in very large databases. In such cases, it may be desirable to estimate the sizes of the query responses, rather than resolve the query itself. Typical approaches to the selectivity estimation problem such as histograms work well in low dimensionality, but degrade rapidly with increasing dimensionality because of dependencies among attributes [11, 13, 27, 28]. For dimensionalities larger than five or six, the error rates become unacceptably high for most real data sets. It has been conjectured in [13] that simple random sampling is likely to outperform most other schemes such as histograms with increasing dimensionality. In this paper, we will show that the local characteristics of the data revealed by the subspace sampling technique can be utilized in order to improve the effectiveness of the selectivity estimation procedure. We will also demonstrate similar results for the approximate nearest neighbor search problem.

In order to facilitate further development of the ideas, we will introduce additional notations and definitions. We assume that the data set is denoted by $\mathcal{D}$. The number of points in the data set is denoted by N and the dimensionality by d. The full dimensional data space is denoted by $\mathcal{U}$. We define the l-dimensional hyperplane $\mathcal{H}(\overline{y}, \mathcal{E})$ by an anchor $\overline{y}$ and a mutually orthogonal set of vectors $\mathcal{E} = \{\overline{e}_1 \ldots \overline{e}_l\}$. The hyperplane passes through $\overline{y}$, and the vectors in $\mathcal{E}$ form the basis system for its subspace. The projection of a point $\overline{x}$ onto this hyperplane is denoted by $\mathcal{P}(\overline{x}, \overline{y}, \mathcal{E})$ and is the closest approximation of $\overline{x}$, which lies on this hyperplane. In order to find the value of $\mathcal{P}(\overline{x}, \overline{y}, \mathcal{E})$, we use $\overline{y}$ as the reference point[1] for the computation. Specifically, we determine the projections of $\overline{x} - \overline{y}$ onto the $\{\overline{e_1} \ldots \overline{e_l}\}$. Then, we translate the resulting point by the reference

[1]We always choose a point on the hyperplane as the reference point.

point $\overline{y}$. Therefore, we have:

$$\mathcal{P}(\overline{x}, \overline{y}, \mathcal{E}) = \overline{y} + \sum_{i=1}^{l} [(\overline{x} - \overline{y}) \cdot \overline{e_i}]\, \overline{e_i} \tag{1}$$

We have illustrated a pictorial representation of $\overline{x'} = \mathcal{P}(\overline{x}, \overline{y}, \mathcal{E})$ in Figure 1. We note that $\overline{x'}$ can be represented in the orthonormal axis system for $\mathcal{E}$ with the use of only l cooordinates $((\overline{x} - \overline{y}) \cdot \overline{e_1} \ldots (\overline{x} - \overline{y}) \cdot \overline{e_l}))$. This incurs the additional overhead of maintaining $\overline{y}$ and $\mathcal{E}$. This is however a constant storage overhead, which can be amortized over the large number of points stored on this hyperplane. The error of approximating $\overline{x}$ with $\mathcal{P}(\overline{x}, \overline{y}, \mathcal{E})$ is given by the Euclidean distance between $\overline{x}$ and $\mathcal{P}(\overline{x}, \overline{y}, \mathcal{E})$ and is denoted by $\Delta(\overline{x}, \overline{y}, \mathcal{E})$. The lossy reduction system discussed in this paper will determine locality specific hyperplanes, so that for each data record, this value is less than a pre-specified tolerance ϵ. In other words, for each data point $\overline{x}$ projected into a hyperplane denoted by $(\overline{y}, \mathcal{E})$, we have $\Delta(\overline{x}, \overline{y}, \mathcal{E}) \leq \epsilon$.

This paper is organized as follows. In the next section, we will introduce the hierarchical subspace sampling technique and discuss some of its properties. In section 3, we will discuss how the data is stored in compressed form using the hierarchically sampled subspaces. Section 4 will discuss the application to the nearest neighbor search and selectivity estimation problems. The empirical results are discussed in section 5. Finally, we present the conclusions and summary in section 6.

1.1 Contributions of this paper

This paper introduces an effective and linearly scalable subspace sampling approach to the problem of data reduction. This technique uses a hierarchical partitioning approach in conjunction with a subspace sampling procedure which is sensitive to the data set at hand. The dual nature of this hierarchical partitioning and subspace sampling approach makes the reduction process very effective. While the subspace sampling approach provides a much more compact representation than traditional dimensionality reduction techniques, it also provides hard bounds on the error of the approximation. A useful property of the subspace sampling technique is that the reduction factor improves with increasing number of database points. While sampling techniques are generally expected to improve with the number of database points, this is not the case for previously proposed subspace sampling procedures [1, 8, 17, 15] which sample dimensions rather than points. The use of a sampling approach also results in a computationally efficient implementation which is almost linearly scalable both with data set size and dimensionality.

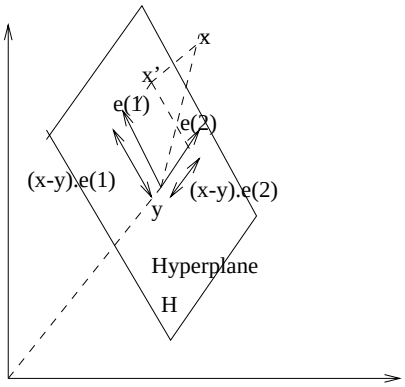


Figure 1: Pictorial Representation of Approximation

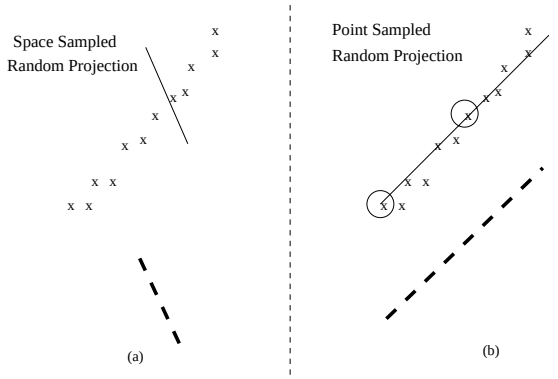


Figure 2: Comparing Point Sampled and Space Sampled Random Projections

We show how to use the subpscae sampling method in order to provide effective solutions to database applications such as nearest neighbor search and selectivity estimation. The subspace sampling method reveals important local characteristics of the data which can be used for effective solutions to these problems. We note that traditional methods for selectivity estimation such as histograms do not provide accurate results for even ten dimensional applications [13], whereas our empirical results indicated that the subspace sampling technique provides accurate results on color-histogram data sets of dimensionality larger than fifty. We will also show that the partitioning created by the hierarchical subspace method can be used for effective nearest neighbor search in a way which is significantly more effective than currently used dimensionality reduction techniques.

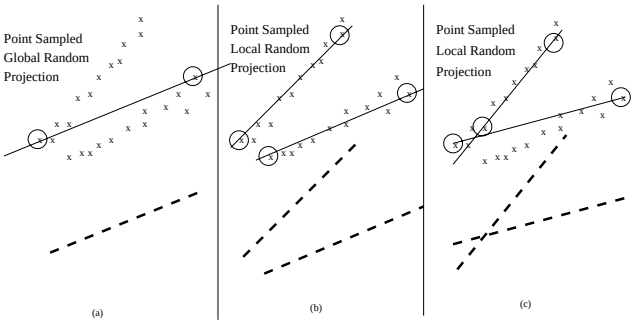


Figure 3: Effects of Data Locality on Subspace Sampling

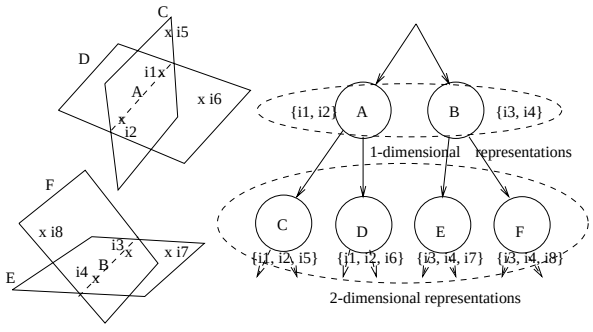


Figure 4: Illustration of the Sampling Procedure

2 The Hierarchical Subspace Sampling Technique

The method of random projections [1, 8, 15, 26] has recently been recognized as an efficient and scalable alternative to dimensionality reduction. These techniques sample[2] spherically symmetric random directions on which the data is projected. Such methods may often require an unnecessarily higher dimensionality to represent the data, since they do not utilize the properties of the particular data set at hand. In order to intuitively understand this point, we will illustrate with the use of 1-dimensional projections of 2-dimensional data. Consider the data set illustrated in Figure 2 in which we have illustrated two kinds of projections. In Figure 2(a), the data *space* is sampled in order to find a 1-dimensional line along which the projection is performed. In data space sampling, random projections are chosen in a spherically symmetric fashion [1] irrespective of the data distribution. The reduced data in this 1-dimensional representation is simply the projection of the data points onto the line, as illustrated in lower diagram of Figure 2(a). Though repeated applications of subspace sampling [1, 8, 17, 15] provide bounds on data reduction quality, it is clear from the above illustration that such a projection may often turn out to be blind to the basic patterns in the data. In the second case of Figure 2(b), we have sampled the *points* in order to create a random projection. The sampled subspace is defined as the $(l-1)$-dimensional hyperplane containing l randomly chosen points from the data. In this case, the chosen subspace is naturally biased by the original data distribution. For example, in Figure 2(b), the 1-dimensional line obtained by sampling two points picks up the directions of greater variance more effectively than the space-sampled random projection of Figure 2(a). For this reason, the quality of the reduction for Figure 2(b) is significantly better than that in Figure 2(a).

While it is intuitively clear that point sampling is more effective than space sampling for variance preservation, the advantages are limited when the data distribution varies considerably with locality. For example, in Figure 3(a), even the optimal 1-dimensional random projection cannot represent all points without losing a substantial amount of variance of the data. In Figures 3(b) and (c), we have used the random projection technique *locally* in conjunction with data partitioning. In this technique, each data point is projected on the closest of a number of point sampled hyperplanes. It is clear that in the latter cases, the projections of the data points onto the lines are the best approximations. This is because each subspace is optimized to a locally sampled set of points. It is

[2]The hyperplanes are repeatedly sampled, and the best alternative is picked in order to determine the final representation.

also interesting to see that even though the data is 2-dimensional, it can be (approximately) represented by projections along 1-dimensional lines. This is because the local implicit dimensionality of each data point is only one, once that data has been partitioned appropriately. We also note that there may be some differences in the quality of the final reduction (such as those in Figures 3(b) and (c)) depending upon the subspaces which get sampled, but the final representation always loses less information than a global approach with the same number of samples. This follows from the straightforward observation that when the k hyperplanes $\mathcal{S}_1 \ldots \mathcal{S}_k$ are sampled, the distance of the data point $\overline{x} \in \mathcal{D}$ to the closest of $\mathcal{S}_1 \ldots \mathcal{S}_k$ is always at most the distance loss of $\overline{x}$ when only one of these hyperplanes (say $\mathcal{S}_i$) is chosen for reduction of each data point in $\mathcal{D}$.

On the other hand, the improvements of the localized subspace sampling technique come at the additional storage costs of the different hyperplanes. This limits the number of hyperplanes which can be retained from the sampling process, and requires us to make judicious choices in picking these hyperplanes. A second important issue is that even the implicit dimensionalities of the different portions of the data partition may be different. Therefore, we need a mechanism by which the sampling process is able to effectively choose hyperplanes of the lowest possible dimensionality for each portion of the data partition. This is an issue which we will discuss after developing some additional notational machinery:

Definition 2.1 *Let $P = (\overline{x_1} \ldots \overline{x_{l+1}})$ be a set of $(l+1)$ linearly independent points. The representative hyperplane $\mathcal{R}(P)$ of P is defined as the l-dimensional hyperplane which passes through each of these $(l+1)$ points.*

The hyperplane $\mathcal{R}(P)$ can also be represented with the use of any point $\overline{y}$ on the hyperplane, and an orthonormal set of vectors $\mathcal{E} = \{\overline{e_1} \ldots \overline{e_l}\}$, which lie on the hyperplane. We shall call $(\overline{y}, \mathcal{E})$ the *axis* representation of the hyperplane, whereas the set P is referred to as the *point* representation. Thus, $\mathcal{R}(P)$ (point representation) is the same as $\mathcal{H}(\overline{y}, \mathcal{E})$ (axis representation). We note that there can be infinitely many point or axis representations of the same hyperplane. The axis representation is more useful for performing distance computations of the hyperplane from individual points in the database, whereas the point representation has advantages in storage efficiency in the context of a hierarchical arrangement of subspaces. We will discuss this issue in greater detail in section 3.

2.1 The Subspace Tree

In this section, we will introduce the subspace tree, which is a conceptual organization of subspaces used in the data reduction process. This conceptual organization imposes a hierarchical arrangement of the subspaces of different dimensionalities. The hierarchical organization is also useful in developing variable dimensionality representations of the data. Each node in the subspace tree corresponds to a hyperplane along with its representative set which is drawn from the database $\mathcal{D}$. The nodes at level-m in the subspace tree correspond[3] to m-dimensional subspaces. The root node corresponds to the null subspace. Thus, the dimensionality of the hyperplane for any node in the tree is determined by its depth. The subspace at a node is hierarchically related to that of its immediate parent. Each subspace other than the null subspace at the root is a 1-dimensional extension of its parent hyperplane. This 1-dimensional extension is obtained by adding a sampled data point to the representative set of the parent hyperplane. In order to elucidate the concept of a subspace tree, we will use an example. In Figure 4, we have illustrated a hierarchically arranged set of subspaces. The figure contains a two-level tree structure which corresponds to 1- and 2-dimensional subspaces. For each level-1 node in the tree, we store two points which correspond to the 1-dimensional line for that node. For each lower level node, we store an additional data point which increases the dimensionality of its parent subspace by 1. Therefore, for a level-m node the representative set is of cardinality $(m+1)$. For example, in the case of Figure 4, the node A in the subspace tree (with representative set $\{i_1, i_2\}$) corresponds to the 1-dimensional line defined by $\{i_1, i_2\}$. This node is extended to a 2-dimensional hyperplane in two possible ways corresponding to the nodes C and D. In each case, an extra point needs to be added to the representative set for creating the 1-dimensional extension. In order to extend to the 2-dimensional hyperplane for node C, we use the point i_5, whereas in order to extend to the hyperplane for node D, we use the point i_6. Note from Figure 4(a) that the intersection of the 2-dimensional hyperplanes C and D is the 1-dimensional line A. The subspace tree is formally defined as follows:

Definition 2.2 *The subspace tree is a hierarchical arrangement of subspaces with the following properties: (1) Nodes at level-m correspond to m-dimensional hyperplanes (2) Nodes at level-$(m+1)$ correspond to hyperplanes which are 1-dimensional extensions of their parent hyperplanes at level-m (3) The point representative set of a level-$(m+1)$ node is obtained by adding a sampled data point to the representative set of its m-dimensional parent subspace.*

[3]We assume that level 0 corresponds to the root.

So far, we have only discussed the concept of a subspace tree, while we have not discussed the algorithmic process of construction. The aim of this paper is to show that by carefully sampling the data points and constructing the subspace tree, each data point can be typically be represented in a relatively low dimensional subspace with very little reconstruction loss.

Once a subspace tree has been constructed, each data point $\overline{x}$ is assigned to a node in this tree, so that the distance of the corresponding hyperplane from $\overline{x}$ is less than the reduction tolerance ϵ. The data point $\overline{x}$ is represented in terms of its coordinates on the hyperplane to which it is assigned. Thus, the amount of space needed to represent $\overline{x}$ depends only on the dimensionality of the corresponding hyperplane rather than the dimensionality of $\mathcal{D}$. Since higher levels of the tree require lower storage overhead, it is desirable to assign $\overline{x}$ to as high a level of the tree as possible. We note the following:

Observation 2.1 *Let S be a set of representative points, and let $\overline{z}$ be a data point which extends the dimensionality of the corresponding hyperplane to that of its child. Let $(\overline{y_1}, \mathcal{E}_1)$ and $(\overline{y_2}, \mathcal{E}_2)$ be the axis representations of S and $S \cup \{\overline{z}\}$ respectively. Then, for any data point $\overline{z}$, it must be true that $||\overline{z} - \mathcal{P}(\overline{z}, \overline{y_2}, \mathcal{E}_2)|| \leq ||\overline{z} - \mathcal{P}(\overline{z}, \overline{y_1}, \mathcal{E}_1)||$.*

The above observation simply states that the distance of the point $\overline{z}$ to a hyperplane $\mathcal{H}(\overline{y_2}, \mathcal{E}_2)$ is lower than the distance to its parent hyperplane $\mathcal{H}(\overline{y_1}, \mathcal{E}_1)$, since the former subsumes the latter. Thus, if the reduced data points are stored in the subspace tree, then as the value of ϵ is reduced, a larger number of points would need to be stored at the lower levels of the tree. Since the storage at lower levels requires a greater number of coordinates for representation, it follows that there is a natural trade-off between the storage requirements and representational accuracy.

2.2 Subspace Tree Construction

In this section, we will show how the subspace tree may be constructed by careful localized sampling of the data points in conjunction with a recursive partitioning of the data. This procedure turns out to be extremely effective in influencing the subspace sampling process so that the resulting subspaces are effectively biased for particular data localities. The input to the subspace sampling algorithm for tree construction is the reduction tolerance parameter ϵ, and the data set $\mathcal{D}$. The subspace tree is constructed hierarchically in top-down fashion, while also partitioning the data set $\mathcal{D}$ along this hierarchy. The subspace tree construction uses a levelwise algorithm in order to

Algorithm *SampleSubspaceTree(CompressionTolerance: ϵ, MaximumTreeDegree: k_{max}, Database: $\mathcal{D}$, Node Limit: L)*
begin
Root = null;
Sample $2 * k_{max} * sampfactor$ points from $\mathcal{D}$;
Pair up points randomly to create $k_{max} * sampfactor$ 1-dimensional point representative hyperplanes (lines) denoted by $\mathcal{S}$;
{ Pick the k_{max} hyperplanes which create a partitioning for which distances between data points and their localized projections are as low as possible }
$\mathcal{S}$ = *SampleBestHyperplanes*($\mathcal{S}, k_{max}$);
{ Let $\mathcal{S}$ have the k_{max} level-1 nodes (hyperplanes) $S_1 \ldots S_{k_{max}}$ }
$(\mathcal{T}(S_1), \ldots \mathcal{T}(S_{k_{max}}), \mathcal{Q}(S_1), \ldots \mathcal{Q}(S_{k_{max}}))$ =*AssignData*($\mathcal{D}, \mathcal{S}$);
{ Remove any node which has fewer than *minthresh* assigned points; These points also become outlier points }
$\mathcal{S}$ = *RemoveNodes*($S_1 \ldots S_{k_{max}}$, *minthresh*); { $\mathcal{L}_m$ is the set of level-m nodes }
$\mathcal{L}_1 = \mathcal{S}$; { Each hyperplane (line) in $\mathcal{S}$ is the child of *Root* };
$m = 1$;
while ($\mathcal{L}_m \neq \{\}$) and (less than L nodes have been generated) **do**
begin
for each level-m node $R \in \mathcal{L}_m$ **do**
begin
Sample $k_{max} * sampfactor$ points from $\mathcal{T}(R)$;
Extend the node R by each of these $k_{max} * sampfactor$ points (in turn) to create the $k_{max} * sampfactor$ corresponding $(m+1)$-dimensional point representative hyperplanes denoted by $\mathcal{S}$;
$\mathcal{S}$ = *SampleBestHyperplanes*($\mathcal{S}, k_{max}$);
$(\mathcal{T}(S_1), \ldots \mathcal{T}(S_{k_{max}}), \mathcal{Q}(S_1), \ldots \mathcal{Q}(S_{k_{max}}))$ = *AssignData*($\mathcal{T}(R), \mathcal{S}$);
$\mathcal{S}$ = *RemoveNodes*($S_1 \ldots S_{k_{max}}$, *minthresh*); { Thus $\mathcal{S}$ contains at most k_{max} children of R }
$\mathcal{L}_{m+1} = \mathcal{L}_{m+1} \cup \mathcal{S}$; m=m+1;
end;
end;
Perform final post-processing phase of reassignment of database $\mathcal{D}$ to nodes in subspace tree;
end

Figure 5: Subspace Tree Construction

build the tree structure. This is done in order to restrict the number of database passes during the tree construction phase. Each node of the subspace tree corresponds to a hyperplane defined by the sequence of representative points sampled, starting from the root up to that node. Therefore, we will be using the term hyperplane and node interchangeably throughout the discussion of the subspace tree.

At each stage of the algorithm, every node N in the subspace tree has a set of *descendent assignments* $\mathcal{T}(N) \subseteq \mathcal{D}$ from the database $\mathcal{D}$. These are the data points which will be assigned to one of the descendants of node N during the tree construction process. In addition, each node also has a set of *direct assignments* $\mathcal{Q}(N)$, which are data points within the specified tolerance ϵ of the hyperplane corresponding to node N. In each iteration, the descendent assignments $\mathcal{T}(N)$ in each of the nodes at a given level are partitioned further into at most k_{max} children of node N. This partitioning is based on the distance of the data points to the hyperplanes corresponding to the k_{max} children of N. Specifically, each data point is assigned to the hyperplane from which it has the least distance. This results in each point from $\mathcal{T}(N)$ becoming either a direct or descendent assignment of one of these k_{max} children depending upon whether or not it lies within the tolerance factor ϵ of the corresponding hyperplane. This process continues until each data point becomes either the direct assignment of some node or is identified as an outlier. The overall algorithm for subspace tree construction is illustrated in Figure 5.

The subspace tree construction algorithm proceeds in an iterative levelwise fashion. The mth level of the tree is constructed during the mth levelwise phase. The reason for this levelwise approach is that the database operations during the construction of a given level of nodes can be consolidated into a single database pass. The actual construction of the mth level is achieved by sampling one representative point for each of the k_{max} children of the level-$(m-1)$ nodes in order to create the corresponding 1-dimensional extension. However, we also use oversampling in order to improve the quality of the resulting subspaces. The subspace sampling algorithm defines a parameter called *sampfactor*, which is the factor by which we oversample the points at a given node from which the final k_{max} representative extensions are chosen. Thus, a total of $k_{max} * sampfactor$ points are picked for extension of the nodes from level-$(m-1)$ to level-m. Only the first iteration of the algorithm ($m = 1$) is special in which we sample $2 * k_{max} * sampfactor$ points in order to create $k_{max} * sampfactor$ lines. Next, the procedure *SampleBestHyperplanes* picks k_{max} lines out of these $k_{max} * sampfactor$ lines for which the localized projection losses are as low as possible. Details

of the *SampleBestHyperplanes* procedure are discussed in subsection 2.3. Once the hyperplanes for the first level nodes have been determined, we assign each point in the database to one of these nodes either as a direct assignment or as a descendent assignment. This is achieved by the procedure *AssignData*, and is discussed in detail in subsection 2.4. We also ensure that those nodes with fewer than *minthresh* points assigned to them are removed from consideration. These are the outlier nodes which are discarded by the procedure *RemoveNodes*. The assigned points for these nodes are outliers which need to be stored separately by the algorithm. Details of this procedure are discussed in subsection 2.5. As a result, the final outdegree of the node may be less than k_{max}. In the example illustrated in Figure 4, the two lines created by this procedure are A and B. The corresponding sampled points which create these lines are $\{i_1, i_2\}$, and $\{i_3, i_4\}$ respectively.

The algorithm then proceeds in a levelwise fashion of building level-m of the tree in the same sequence of operations as discussed above for level-1 of the tree. The main difference for $m \geq 2$ is in the methodology for extending the subspaces by a dimensionality of one. In this case, for each node N, we sample $k_{max} * sampfactor$ points from $\mathcal{T}(N)$. The process of sampling the points from $\mathcal{T}(N)$ intentionally biases the children subspaces depending upon the data distribution of $\mathcal{T}(N)$. Further, the purpose of oversampling by a factor of *sampfactor* is to increase the effectiveness of the final children subspaces which are picked. The larger the value of *sampfactor*, the better the sampled subspaces, but the greater the computational requirement. Thus, a total of $k_{max} * sampfactor$ m-dimensional hyperplanes can be generated by combining the representative points from node N with each of these sampled points. In each iteration, the algorithm assigns the data points in a given node N to its closest child. Next, the *SampleBestHyperplanes* procedure picks the k_{max} hyperplanes out of these $k_{max} * sampfactor$ hyperplanes in order to create the most effective partitioning. As in the case of level-1 nodes, the *AssignData* procedure determines the assignments of the data points in the nodes of $\mathcal{T}(N)$ to the respective children. In the example illustrated by Figure 4, the 1-dimensional hyperplane A is extended to the 2-dimensional hyperplanes C and D by adding the points i_5 and i_6 respectively to the representative set of A.

We note that in Figure 5, we have presented the *AssignData* procedure separately for each node for ease in description. In the actual implementation, this procedure is executed simultaneously for all nodes at a given level in one scan. Similarly, the process of picking the best hyperplanes for all nodes at a given level is executed simultaneously in a single scan of the data. We will discuss details of these issues in a later subsection. The process of levelwise tree construction continues until no

node in the current level can be extended any further, or the maximum limit L for the number of nodes has been reached. This limit L is governed by the amount of available memory since we would like the subspace tree to be memory-resident for a number of useful applications such as nearest neighbor indexing and selectivity estimation. For our implementation, we used a conservative limit of only $L = 10,000$ nodes, which was well within current main memory limitations for even 1000-dimensional data sets. At the end of the subspace tree construction process, we re-optimized the assignment of each data point $\overline{x}$ by finding the hyperplane at the highest level of the tree for which the distance value was less than ϵ. In many cases, this reduces the data even further by reducing the dimensionality of the representation.

Each of the procedures *SampleBestHyperplanes* and *AssignData* require the computation of distances of data points $\overline{x}$ to the representative hyperplanes. In order to perform these distance computations, the axis representations of the hyperplanes need to be determined. A hyperplane node N at level-m is only implicitly defined by the $(m+1)$ data points $\{\overline{z_1} \ldots \overline{z_{m+1}}\}$ stored at the nodes along the path from the root to N. The next tricky issue is to compute the axis representation $(\overline{y}, \mathcal{E} = \{\overline{e_1} \ldots \overline{e_m}\})$ of the points $\{\overline{z_1} \ldots \overline{z_{m+1}}\}$ efficiently in a way that can be replicated exactly at the time of data reconstruction. This is especially important, since there can be an infinite number of axis representations of the same hyperplane, but the projection coordinates are computed only with respect to a particular axis-representation. The corresponding representation $(\overline{y}, \mathcal{E} = \{\overline{e_1} \ldots \overline{e_m}\})$ is computed as follows:

We first set $\overline{y} = \overline{z_1}$ and $\overline{e_1} = (\overline{z_2} - \overline{z_1})/||\overline{z_2} - \overline{z_1}||$. Next, we iteratively compute $\overline{e_i}$ from $\overline{e_1} \ldots \overline{e_{i-1}}$ as follows:

$$e_i = \frac{\overline{z_{i+1}} - \overline{z_1} - \sum_{j=1}^{i-1} [(\overline{z_{i+1}} - \overline{z_1}) \cdot \overline{e_j}] \, \overline{e_j}}{||\overline{z_{i+1}} - \overline{z_1} - \sum_{j=1}^{i-1} [(\overline{z_{i+1}} - \overline{z_1}) \cdot \overline{e_j}] \, \overline{e_j}||} \tag{2}$$

The set $(\overline{y}, \mathcal{E})$ generated by the Equation 2 is an axis representation of the hyperplane defined by the points $\{\overline{z_1} \ldots \overline{z_{m+1}}\}$. This is because this technique is essentially a modified version of the Gram Schmidt orthogonalization process.

Many axis representations can be generated using Equation 2 for the same hyperplane $\mathcal{R}(\{\overline{z_1} \ldots \overline{z_{m+1}}\})$ depending upon the ordering of $\{\overline{z_1} \ldots \overline{z_{m+1}}\}$. Since we need to convert from point representations to axis representations in a consistent way for both data reduction and reconstruction, this ordering needs to be fixed in advance. For the purpose of this paper, we will assume that the point ordering is always the same as one in which it was sampled during the top-down tree construction process. This leads to representative points sampled at higher levels of the tree to be ordered first, and points

at lower levels to be ordered last. The only ambiguity is for the level-1 nodes at which 2 points are stored instead of one. In that case, the record which is lexicographically smaller is ordered earlier. We shall refer to this particular convention for axis representation as the *path-ordered axis representation*.

2.3 Oversampling and Selection of Subspaces

In this section, we will discuss the details of the *SampleBestHyperplanes* subroutine of Figure 5. The subspace sampling procedure oversamples the number of hyperplanes by a factor of *sampfactor*. This is done in order to improve the quality of the subspaces determined. The first task is to partition the $k_{max} * sampfactor$ hyperplanes into $sampfactor$ sets of k_{max} hyperplanes. We will pick one of these partitions depending upon the quality of the assignment of the data points to these hyperplanes. In order to do this, the distance of the data point $\overline{x}$ to each of the $k_{max} * sampfactor$ hyperplanes is determined. For each of the $sampfactor$ sets of hyperplanes, we assign the data point $\overline{x}$ to the closest hyperplane from that partition. This results in a total of $sampfactor$ possible assignments of the data points. The cost of the assignment is the average distance of the data point to its assigned hyperplane, and is equal to the average distance information lost by the corresponding reduced representation. The lowest cost of these *sampfactor* assignments is determined. The *SampleBestHyperplanes* procedure returns the k_{max} points which can be used to extend the current node to each of the k_{max} children by a dimensionality of one. We note that the *SampleBestHyperplanes* procedure may not necessarily provide an optimum solution, but turns out to be effective in practice because of its simplicity and implementation efficiency.

2.4 Partitioning the Points

In this section, we will describe the *AssignData* procedure of Figure 5. The procedure *AssignData* partitions the points among the children nodes, and also decides whether the assignment of a data point $\overline{x}$ to a hyperplane is of the descendent or direct type. For each child hyperplane $(\overline{y}, \mathcal{E})$, the distance value $\Delta(\overline{x}, \overline{y}, \mathcal{E})$ is calculated. Next, we check if this value is below the compression tolerance ϵ. If so, then the data point $\overline{x}$ is directly assigned to that node. Otherwise, it is assumed that a higher implicit dimensionality is needed to represent that point and it is considered a descendent assignment. The top-down algorithmic process of subspace tree construction ensures that such a data point will be a direct assignment for one of the descendants of its current node,

unless it is determined to be an outlier.

2.5 Removal of Outlier Nodes

This procedure is denoted by *RemoveNodes* in Figure 5. Many points in any data set may be outliers for which efficient locality specific representations cannot be found. Such points need to be stored separately by the algorithm. In each iteration, we find all nodes in the current level of the tree which have less than *minthresh* descendent assignments. These nodes and the corresponding points are removed by the algorithm. The corresponding points are stored separately in their full dimensional representation.

We note that the tree structure may be optimized a variety of ways by picking different outdegrees at various levels of the tree. Finding this optimal structure is a complex theoretical problem which is outside the scope of this paper. Our primary aim is to show that a simple binary tree implementation of the subspace tree structure can yield substantial advantages over global dimensionality reduction methods. This illustrates the power of the simple concepts underlying the technique.

2.6 Disk Sensitive Implementation for Large Databases

Each of the procedures *AssignData* and *SampleBestHyperplanes* require the assignment of the data to nodes at a given level of the tree. In order to improve the I/O efficiency, we process all the nodes at a given level in a single database scan. We maintain an additional vector with one entry for each database record. Each entry in this vector indicates the node to which the corresponding database record is assigned and whether the corresponding record is a descendent or direct assignment. During the database scan, we use the vector to find the hyperplane $(\overline{y}, \mathcal{E})$ for each database record $\overline{x}$. This is then used to calculate the value of $\Delta(\overline{x}, \overline{y}, \mathcal{E})$ for the *AssignData* and *SampleBestHyperplanes* procedure.

3 Storage and Reconstruction

Since the reduction process stores the reduced data in the context of a hierarchical subspace tree structure, we need to maintain the following two pieces of information:

- **The Subspace Tree:** This is a constant overhead which can be maintained very efficiently with the use of the *point representation*. For each level-1 node we maintain the two points which define the sampled line in lexicographic ordering. For each level-m node, we maintain the additional data point which increases the dimensionality of the corresponding subspace by one. In addition, we need to maintain the identity of the node and its immediate parent, which requires another two integers for each node. Thus, for a subspace tree with β nodes, the storage requirement is of the order of $(\beta \cdot (d+2))$ values. This is almost the best that one could hope to achieve, since at least $\beta \cdot d$ values will always be required in order to store all the β subspaces of a d-dimensional space. In fact if the subspaces were maintained explicitly, then the storage requirement would be $\beta \cdot d \cdot l$ values for an average subspace dimensionality of l. This is because explicit storage of the axes require maintenance of a d-dimensional vector for each of the l axes of β nodes. The storage requirement of $\beta \cdot (d+2)$ values requires the storage of only 1 additional vector for each of the β subspaces. The reason for this extraordinarily high storage efficiency is the use of the point representation in which the hyperplane at a given node is not stored explicitly, but is implicitly represented by the points stored along the path from that node to the root. (Therefore, the vector stored at a node is reused for the subspace representation of all descendents of that node.) Our empirical results indicated that the overhead for maintaining the subspace tree is very small compared to the storage requirements of the database itself.

- **The Reduced Database:** For each data point, we need to maintain one integer which indicates the identity of the node for which it is a direct assignment. In addition, we maintain the coordinates of the data point for the axis representation $(\overline{y}, \mathcal{E})$ of this hyperplane in accordance with Equation 1. For the data point $\overline{x}$, these coordinates are given by $(c_1 \ldots c_m) = \{\overline{e_1} \cdot (\overline{x} - \overline{y}) \ldots \overline{e_m} \cdot (\overline{x} - \overline{y})\}$. Thus, only $(m+1)$ values need to be stored for each database point.

3.1 Reconstruction Algorithm

The reconstruction algorithm proceeds in two phases. In the first phase, the (path-ordered) axis representation of the subspace tree is built. In the second phase, this subspace tree is used in order to reconstruct the database.

At first sight, it would seem that the first phase could be time consuming, since for each node in

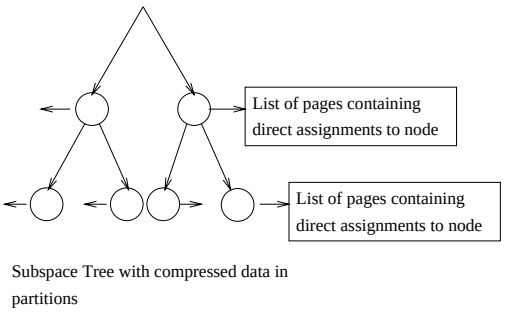


Figure 6: The Subspace Index

the subspace-tree, we would need to find its d-dimensional axis representation. However, it turns out that because of the use of the path-ordered convention for axis-representations, the first phase can be achieved in a time complexity which requires the computation of only one axis per node. The trick is to construct the axis representations of the nodes in the tree in a top-down fashion. This is because the axis representation $\{\overline{e_1} \ldots \overline{e_i}\}$ of a given node can be computed using the axis representation $\{\overline{e_1} \ldots \overline{e_{i-1}}\}$ of its parent and the point $\overline{z'}$ stored at that node in just the single computation of Equation 2. (For the nodes at level-1, lexicographic ordering of the representative points is assumed.) This also automatically results in the path-ordered axis representation of the node.

Once the axis representations of the nodes have been constructed, it is simple to perform the necessary axis transformations which represent the reconstructed database in terms of the original attributes. Recall that for each database point $\overline{x}$, the identity of the corresponding node is also stored along with it. Let $(\overline{y}, \mathcal{E})$ be the corresponding hyperplane and $(c_1 \ldots c_m) = \{\overline{e_1} \cdot (\overline{x} - \overline{y}) \ldots \overline{e_m} \cdot (\overline{x} - \overline{y})\}$ be the coordinates of $\overline{x}$ along this m-dimensional axis representation. Then, as evident from Equation 1, the reconstructed point $\overline{x'}$ is given by:

$$\overline{x'} = \overline{y} + \sum_{i=1}^{m} [c_i] \, \overline{e_i} \tag{3}$$

4 Applications to Approximate Nearest Neighbor Search and Selectivity Estimation

The hierarchical subspace sampling techniques discussed in this paper have the property that they do not treat all parts of a high dimensional data set equally; some of the parts can be represented using a small number of dimensions, whereas other parts are inherently high dimensional. Traditional indexing structures [14, 20, 21, 25] and selectivity estimation techniques [27, 28] treat all parts

of the data in a homogeneous way from the dimensionality perspective; a strategy which results in the worst-case behavior of the data to dominate. In this section, we will provide additional insights into how hierarchical subspace sampling techniques can lead to inherently more effective solutions to such problems by using its variable dimensionality local decompositions. The aim of writing this section is to demonstrate that even simple applications of the proposed principles can lead to dramatically improved solutions for difficult high dimensional problems. A detailed treatment of the optimization of these schemes will be discussed in future work.

4.1 Application to Approximate Nearest Neighbor Search

In the approximate nearest neighbor search problem, we would like to find the nearest neighbor to a given target record within the pre-specified error bound of ϵ. Since our compression system provides such a worst-case guarantee, a sequential scan can be used on the reduced representation in order to find the approximate nearest neighbor. In fact, any compression method can provide savings in I/O even with the use of a sequential scan, as long as the reconstruction procedure can recognize individual records as they are generated in main memory. Such a technique works quite effectively, since the I/O requirements are the motivation for index structure construction. However, the subspace tree provides savings even beyond the advantages of lower storage requirements, since it allows us to create an index in which large portions of the *reduced representation* need not even be accessed. The ability to use such query optimization directly on compressed database systems has recently been recognized as a promising approach for optimizing performance in database systems.

The subspace tree imposes a natural partitioning of the data in which similar records occur together in one block. Unlike an index tree in which only leaf nodes contain the individual records, we allow each node in the tree to point to a *list* of pages which contain all the direct assignments to that node. Thus, the internal tree size is only dependent upon the original subspace tree, rather than the restrictions created by the page sizes of individual nodes. Furthermore, the subspace tree construction algorithm imposes a maximum limit L on the number of subspace tree nodes, which is determined by the main memory limitations. Thus, the subspace tree itself is maintained in main memory. On the other hand, the lists of pages pointed to by each node are maintained on the disk. We also note that since each node points to multiple pages, and only the last of these pages is underfull, this does not lead to s significant reduction in the efficiency of representation for search purposes. The index structure is illustrated in Figure 6.

The actual nearest neighbor search of the tree uses a branch and bound method on the partitioning created by the direct assignments of data points to nodes. The branch and bound method is a classical technique in combinatorial optimization. It uses an ordered search method on a partitioning of the data in order to avoid searching many of the sets in this partitioning. A global pessimistic bound is maintained which provides an upper bound on the distance of the query point to the nearest neighbor. Pruning is done by finding good *optimistic* bounds (lower bounds) on the distance of a target point to each set in this partition. A set may be pruned when its optimistic bound is higher than the global pessimistic bound. For example, in the case of a query point $\overline{q}$ and subspace $(\overline{y}, \mathcal{E})$, this optimistic bound for any direct assignment of that hyperplane is given by $||\overline{q} - \mathcal{P}(\overline{q}, \overline{y}, \mathcal{E})||$. This is the nearest distance between the query point and the hyperplane, and any point lying on the hyperplane cannot have distance lower than this value. The global pessimistic bound is the nearest distance to any subset of the data accessed so far. However, unlike a traditional branch and bound technique, we cannot prune entire subtrees by using the optimistic bound at the root of the subtree. This is because in traditional index structures, lower level nodes are subsumed by higher level nodes, whereas in the subspace tree structure, the lower level subspaces subsume the higher level subspaces. Therefore, in a traditional index structure, the optimistic distance bounds from the query point to the nodes along a given path increase with the depth of the tree structure, whereas this is the reverse in a subspace tree (see Observation 2.1). Hence, one cannot use the optimistic bounds at the root of a subtree as representative of the optimistic bounds at the lower level nodes. In order to account for this, we treat each node as an independent entity irrespective of the hierarchical relationships between the nodes. Initially the pessimistic bound is set to the closest of all the outlier points and is gradually updated as more and more records are accessed. The nodes in the tree are accessed in increasing order of optimistic distance. When the distance $||\overline{q} - \mathcal{P}(\overline{q}, \overline{y}, \mathcal{E})||$ is larger than the pessimistic bound, all data points assigned to this node can be pruned from consideration. This is because it is certain that all data points which lie on this hyperplane are not as close as the best point found so far. Otherwise, we need to access the records in the list for the node $(\overline{y}, \mathcal{E})$, and calculate their distances to the query point q. This may result in the improvement of the pessimistic bound to the closest record to query point q lying on this hyperplane. The record corresponding to the pessimistic bound at termination is returned as the approximate nearest neighbor. We note that this approximate nearest neighbor is an exact nearest neighbor over the set of reduced data points. However, since each reduced data point may have distance at most ϵ from its original representation, it follows that the final nearest neighbor

found lies within the error tolerance of ϵ. The method can also be easily extended to a generalized k-nearest neighbor search problem. For this case, we maintain the global pessimistic bound as the kth best data point found so far by the search procedure. A search procedure is applied to the index structure as in the previous case. A data node is pruned when the optimistic bound for that node is worse than the global pessimistic bound. We note that this method shows the property that the (found) k-nearest neighbor lies within an error bound of the (true) k-nearest neighbor by a distance of ϵ. The correctness of this procedure follows a similar logic as that of finding a single nearest neighbor.

4.2 Application to Selectivity Estimation

The selectivity estimation problem is a difficult one for high dimensional data because of the sparsity of the points [11, 13, 27, 28]. For the high dimensional case, it has been conjectured from empirical evidence [13] that simple random sampling may be the most effective method for selectivity estimation. Specifically, the work in [13] states: "We conjecture that sampling will outperform any of these techniques for dimensionality of around 10, but the error will be too large to make the technique practical."

The reason for the (relative) robustness of random sampling in higher dimensionality is that it can model the correlations in multi-dimensional data more accurately than methods such as histograms, since the correlations in the data are also reflected in the sample. On the other hand, the histogram technique is significantly more effective for lower dimensional cases.

The subspace tree procedure naturally reveals those parts of the data which have low implicit dimensionality. Since it is known that different techniques work more effectively in different implicit dimensionalities, this data decomposition naturally suggests an *ensemble-based* approach to the problem. Ensemble based approaches have recently become quite popular in methods such as indexing and classification [6, 19] because of their ability to combine different techniques in a flexible way so that the final solution is significantly more robust than the use of each individual method.

In the ensemble-based approach for selectivity estimation, we use a histogram based technique for all hyperplanes in the subspace tree with dimensionality at most q_{max}, whereas we use random sampling for data points in the higher dimensional components. The histograms for the lower dimensional component are built *directly* on the sampled hyperplanes, and are thus not parallel

to the original axis-system. Thus, a separate set of histograms is constructed for each partition of the data in its locally optimized subspace. A wide array of methods are available to construct histograms. This choice is orthogonal to our primary aim of showing the effectiveness of the decomposition and the corresponding ensemble based approach. We will demonstrate that even the use of a simple equi-width grid based histogram for the lower dimensional component of the ensemble is sufficient to outperform existing approaches.

Let n_s be the total number of points with implicit dimensionality larger than q_{max} and n_t be the remaining number of points. We define a fraction r_f known as the *representation factor*, which has the same value but is defined differently for each of the two components of the ensemble:
(1) For the random sampling approach, the representation factor r_f is defined as the fraction of the n_s points (with implicit dimensionality greater than q_{max}) which are sampled.
(2) For the histogram based approach, the representation factor r_f is defined as the number of buckets b used divided by the number n_t of low implicit dimensionality points.

For the histogram based approach, we assume uniform distribution within each bucket and store the number of points and index of each bucket. Empty buckets are not included in this list. The index of a bucket provides its position in the multidimensional grid assuming a unique ordering convention of the grid points. Thus, only two values are required in order to store a bucket, while d values are required to store each sampled point. In addition, a fixed amount of space C_T is required in order to store (the relevant segment of) the subspace tree. Thus, the total space requirement for this ensemble based procedure is given by $C_T + r_f \cdot (n_t \cdot 2 + n_s \cdot d)$. We note that C_T is likely to be (asymptotically) negligible for very large databases. The n_t histogram buckets are divided among the different hyperplanes in the subspace tree. The number of buckets assigned to each hyperplane is proportional to the number of points assigned to it. Therefore, for a m-dimensional hyperplane with q assigned buckets, the number of intervals into which the data is discretized is given by $\lfloor q^{1/m} \rfloor$. This also results in adaptive bucket sizes and orientations depending upon the subspace specific data localities.

At query time, we determine all the buckets which intersect with user-specified query ranges. Let the total number of (extrapolated) points[4] in these buckets be denoted by s_1. Similarly, for the

[4]For partially intersected buckets, we needed to find the fraction of the bucket inside user-specified ranges. In most cases, buckets were intersected by only one of the user specified constraints, in which case extrapolation was straightforward for 2- or 3-dimensional buckets. We included or omitted the entire count of buckets which were either greater than 3-dimensional (for $q_{max} > 3$) or intersected with more than one constraint. This was determined by whether or not the bucket center lay inside the user specified range.

sampling component of the ensemble, we determine the total number of points s_2 from the random sample which lie in the user specified ranges. Then, the expected number of points from the second component of the ensemble was given by s_2/r_f. Therefore, the total selectivity estimated by the ensemble is given by $s_1 + s_2/r_f$. The histogram component can be significantly improved with the use of more sophisticated methods such as those in [13] though our aim in this section is to only show the effectiveness of the ensemble-based subspace decomposition principle. In the next section, we will show that this approach achieves much greater accuracy over random sampling for difficult high dimensional cases.

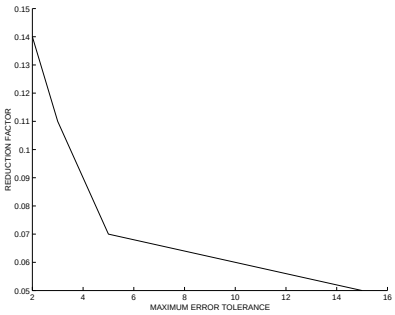
5 Empirical Results

The system was implemented on an AIX 4.1.4 system with 233 MHz and 100 MB of main memory. The data was stored on a 2GB SCSI drive. The breadth and scope of our empirical results included not just the effectiveness anf efficiency of the compression system, but also the effectiveness of the subspace sampling technique to the nearest neighbor and selectivity estimation problems. We tested the subspace sampling method for the following measures: (1) Effectiveness of data reduction. (2) Efficiency of data reduction. (3) Effectiveness of indexing directly on the reduced subspace tree representation. (4) Effectiveness of the ensemble based method for selectivity estimation.

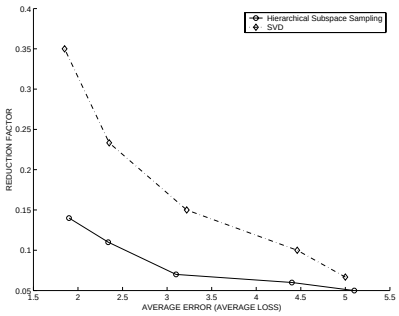
In each case, we will show that the hierarchical subspace sampling method achieves considerable improvements over currently used methods for data reduction in terms of efficiency, effectiveness, or both. In addition, we will show that the hierarchical subspace sampling technique achieves orders of magnitude improvement over standard random sampling techniques which are used for selectivity estimation. Unless otherwise mentioned, the parametric values $k_{max} = minthresh = 2$, and $sampfactor = 10$ were used in the implementation.

5.1 Performance of data reduction

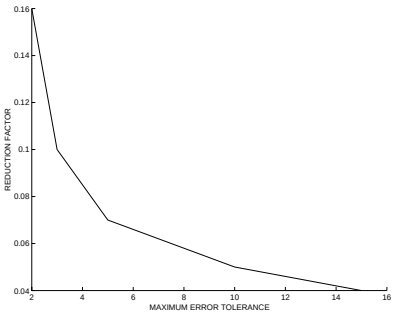
For the purpose of testing, we used a combination of synthetic and real data sets. The synthetic data sets each contained $n_c = 20$ clusters which were gaussian in nature. The (relative) number of points in each cluster was determined by generating a uniform random number between 0 and 1. The centroids of the clusters were chosen randomly. The axis system of each gaussian cluster was arbitrarily oriented with respect to the original data set and the radius along each axis was



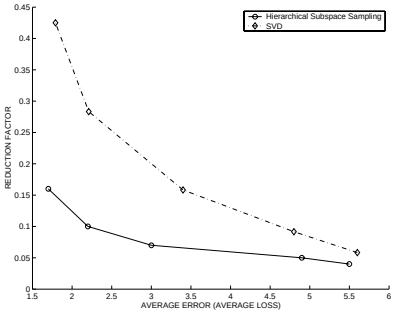
(a) Error Tolerance vs. Red. Factor (Syn. 1)



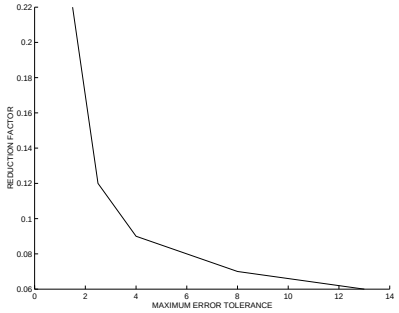
(b) Average Loss vs. Red. Factor (Syn. 1)



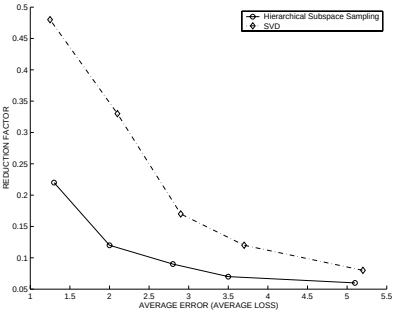
(c) Error Tolerance vs. vs. Red. Factor (Syn. 2)



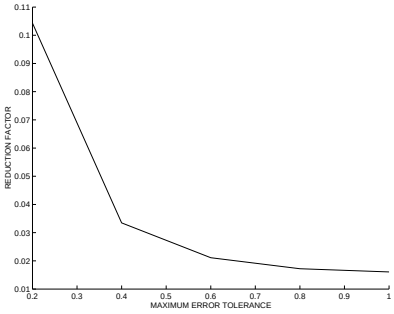
(d) Average Loss vs. Red. Factor (Syn. 2(



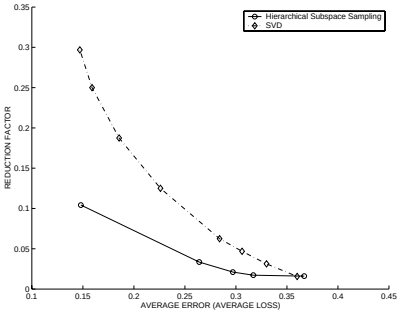
(e) Error Tolerance vs. Red. Factor (Syn 3)



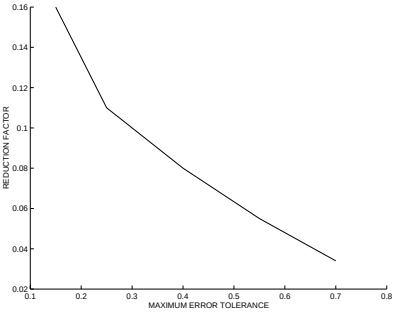
(f) Average Loss vs. vs. Red. Factor (Syn 3)



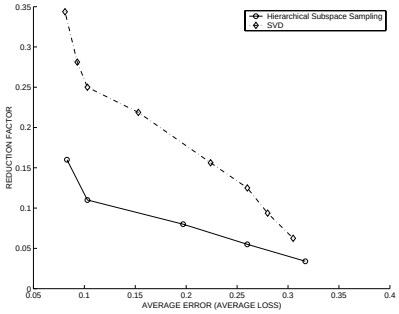
(g) Error Tolerance vs. Red. Factor (64d hist)



(h) Average Loss vs. vs. Red. Factor (64d hist)



(i) Error Tolerance vs. Red. Factor (32d hist)



(j) Average Loss vs. Red. Factor (32d hist)

Figure 7: Average Loss and Error Tolerance for Synthetic and Real Data Sets

chosen as an exponential distribution with 1% of the average distance between the clusters along the individual dimensions. We generated three data sets with dimensionalities 150, 200 and 250 respectively. Each data set contained 100,000 records. We shall refer to these data sets as synthetic data sets 1, 2, and 3 respectively. We also used a 32-d and a 64-d color-histogram data set for testing purposes. In Figures 7(a), (c), (e), (g), and (i) we have illustrated the effect of the error tolerance threshold on the reduction factor on each of the five data sets. The reduction factor was defined as the fraction of the original data set size occupied by the reduced representation (including the subspace tree itself.) In each case, the compression was more effective at higher tolerance levels. This is because of the natural tradeoff between reduction quality and compactness. In order to get a better understanding of the effectiveness of the technique, we compared it to the Singular Value Decomposition method.[5] We compared the reduction factor of the our technique with SVD using the average loss on the X-axis. The average loss was defined as the average distance between the original record and the projected record in the reduced representation. In each case of Figures 7(b), (d), (f),(h), and (j) we have plotted the average reduction factor versus the average loss of each record for both methods. It is clear that the subspace sampling technique is significantly more effective than the standard dimensionality reduction technique. We note that since SVD provides optimal results for the global case, it shows that our technique is at least able to outperform any global reduction algorithm. Another interesting observation from the charts is that the relative compression performance of the subspace sampling technique improves with reduced error tolerances. This is because for very relaxed (high) loss rates, it suffices to represent the data in 1- or 2-dimensional format for either of the two methods. As the error tolerances are tightened, the advantages of localized subspace sampling begin to show up, and the method is able to represent the data in a much smaller number of dimensions. As a result, the overall space required by the subspace tree representation is significantly lower than the standard dimensionality reduction method. Furthermore, we note that these improved results are in spite of the fact that the subspace sampling method provides hard guarantees on the error tolerances, whereas this is not achieved by the standard dimensionality reduction method. It is evident by comparing[6] the different charts in Figure 7 that in each case the average loss was about 50% of the error tolerance for the subspace sampling method. For each case, we also recorded the total number of nodes

[5]The SVD implementation computed only the appropriate number of eigenvectors as was required for the reduction process.

[6]For example, by matching the common axis value (reduction factor) of Figure 7(a) and (b), one can obtain the relationship between the error tolerance and average loss. We have omitted the corresponding explicit graphs for lack of space.

Data set	Tree Nodes
Syn1	78
Syn2	85
Syn3	107
Histogram (64d)	51
Histogram (32d)	56

Table 1: Number of Nodes in Subspace Tree for Each Data Set

$sampfactor$ (Compression Factor)	$minthresh$ (Compression Factor)
2 (0.19)	1 (0.18)
5 (0.15)	2 (0.14)
10 (0.14)	4 (0.16)
20 (0.13)	8 (0.19)

Table 2: Sensitivity to Parameter Values

in the subspace tree structure for the highest tolerance constraint on the number of nodes. This corresponds to the left hand side of each axis in Figures 7(a), (c), (e), (g), and (i) respectively. The results are illustrated in Table 1. As illustrated, less than 150 nodes were required in each case. We have also illustrated the sensitivity of the algorithm to different parameters in Table 2 for the case synthetic data set 1. For each parameter value, the corresponding reduction factor is indicated in brackets. For the case of the sampling factor, it is clear that the the performance of the algorithm improves considerably initially, but tapers off with larger values of the $sampfactor$. The default value of $sampfactor = 10$ was chosen as an efficient solution throughout this paper. We note that the running time increases linearly with $sampfactor$. Therefore, at the expense of some more computational cost, it is possible to improve the efficiency of the scheme further. In Table 2, we have also illustrated the sensitivity of the scheme to $minthresh$. In this case, it is clear that a choice of $minthresh$ which is either too large or too small leads to inefficiency. This is because in one case, too many nodes are created. In the other case, too many points get classified as outliers.

We also tested the performance of the hierarchical subspace sampling technique with increasing database size. In Figures 8 and 9, we have illustrated the behavior of the reduction factor with database generations of different sizes for the parameters of the synthetic data sets Syn. 1 and Syn. 3. On the X-axis, we have illustrated the database size in records whereas the reduction factor is illustrated on the Y-axis. The maximum error tolerance was kept constant at 2% of the standard

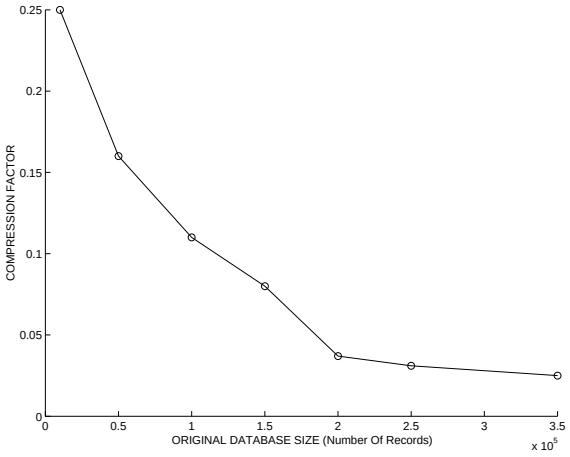


Figure 8: Improvement in Reduction Efficiency with Increasing Database Size (Synthetic 1)

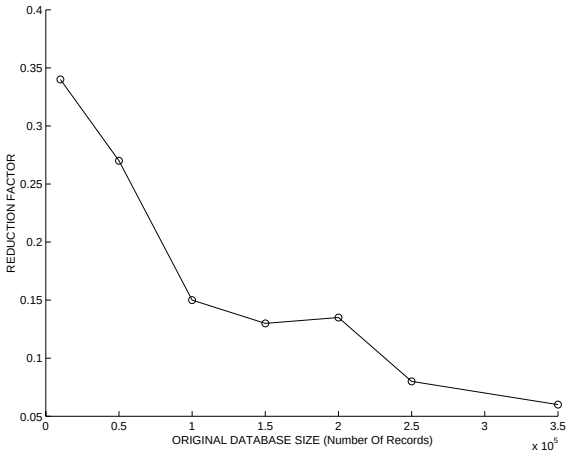


Figure 9: Improvement in Reduction Factor with Increasing Database Size (Synthetic 3)

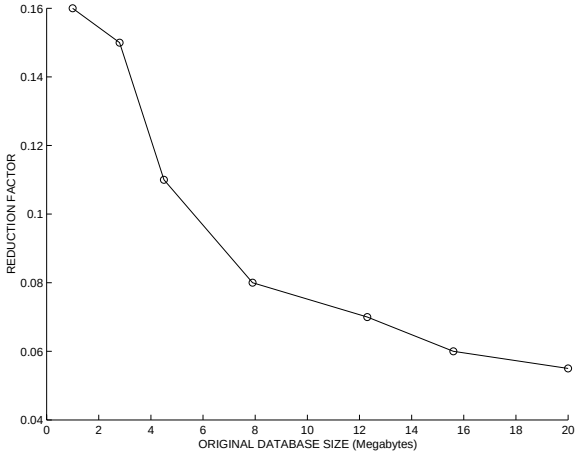


Figure 10: Improvement in Reduction Efficiency with Increasing Database Size (Color Histogram 32-d)

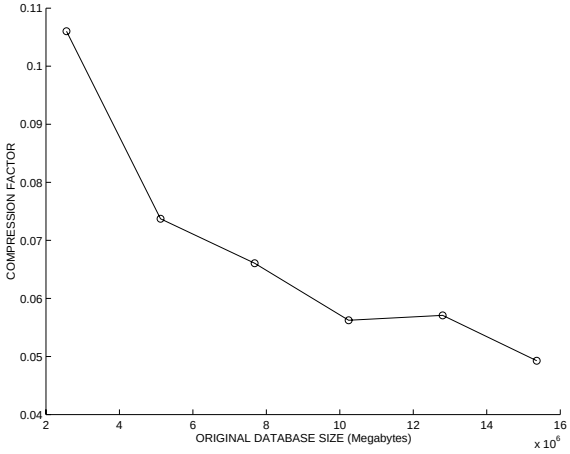


Figure 11: Improvement in Reduction Factor with Increasing Database Size (Color Histogram 64-d)

deviation of the data. It is clear that the reduction factor improves with increasing database size. In Figure 10, we have illustrated the reduction factor with increasing number of records for the case of the 32-dimensional color histogram data set. In this case, the error tolerance ϵ was set at 0.23. In Figure 11, we have illustrated the same results by sampling an increasing number of records from the 64-dimensional color histogram data set. In this case, the error tolerance was set at $\epsilon = 0.3$. Thus, the synthetic and real data sets show very similar trends. It was our experience over a number of data sets that the *reduction factor always improved with increasing database size.* This is a very useful property of the subspace sampling technique, since the data reduction problem is motivated by the large size of data sets. There are two reasons for this behavior:

(1) The size of the subspace tree itself scales sublinearly with database size. In fact, for most data sets that we tested, the subspace tree size increased only marginally for database sizes above 100,000 points. At this point, all the major subspace patterns are already significantly represented in the tree structure as well as the database. In all cases, the total number of nodes in the subspace tree was only about 0.5-10% of the maximum limit $L = 10,000$ nodes.
(2) For larger data sets, the local subspaces determined by the sampling technique are more refined. These refined nodes are reflected in the lower levels of the subspace tree. As a result, a large number of points which would otherwise get classified as outliers are reflected in some lower dimensional projection in the subspace tree. The basic intuition is that in larger data sets, all the natural local data patterns can be reflected in a refined way, which leads to a more optimized representation.

5.2 Efficiency of Data Reduction

We tested the efficiency of the scheme for dimensionality reduction. In order to test the efficiency, we need a data set in which the dimensionality can be varied effectively, while retaining the basic structure of the data. To this effect, we found the market basket data generator of [4] useful. We derived[7] two data sets from the data sets T20.I20.D100K and T15.I15.D100K respectively. This was done by using random projections of varying dimensionality. The error tolerance was fixed at 5% of the standard deviation in each case. In Figures 12 and 14, we have illustrated the scalability of the approach with increasing data dimensionality. It is clear that for lower dimensionalities, the standard SVD approach performs more effectively, but for dimensionalities higher than 60 to 80,

[7]We are using a procedure described in detail in [4]. The transactions are generated by pre-generating a set of potentially frequent baskets and then combining these potentially frequent baskets in order to create transactions. For the transaction Tx.Iy.Dz, it is assumed that each transaction contains x items, the average size of the frequent basket is y, and the number of data points is given by z.

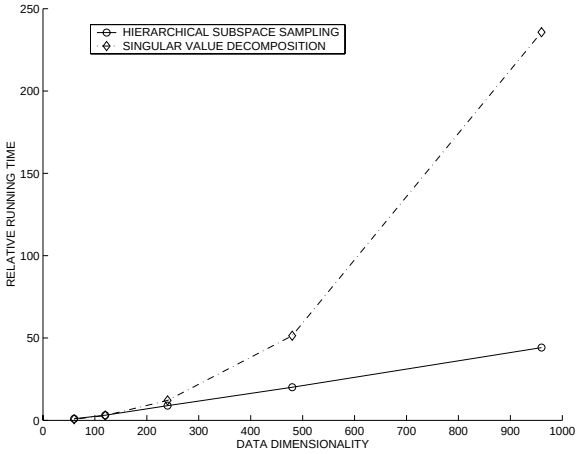


Figure 12: Efficiency of Reduction with Increasing Dimensionality (projection of T20.I20.D100K)

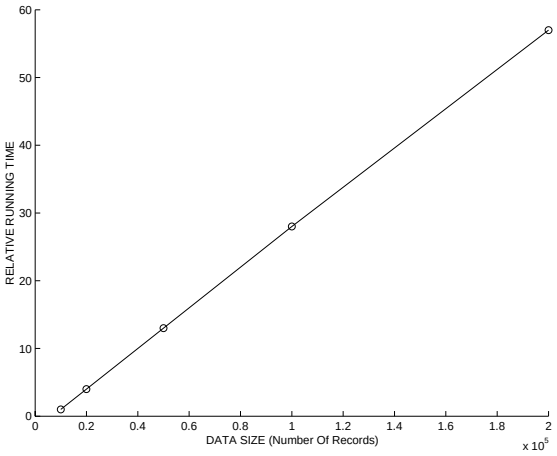


Figure 13: Efficiency of Reduction with Increasing Database Size (T20.I20.Dx)

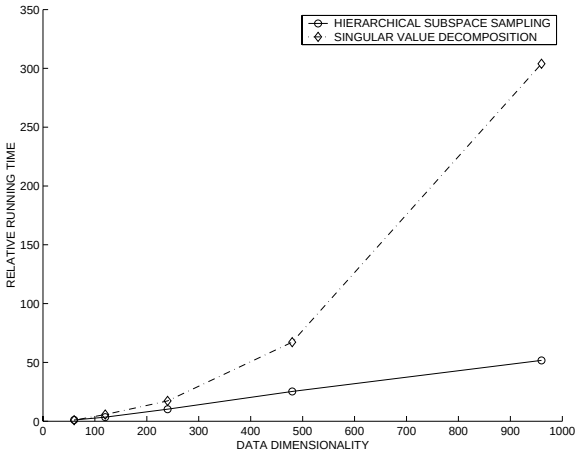


Figure 14: Efficiency of Reduction with Increasing Dimensionality (projection of T15.I15.D100K)

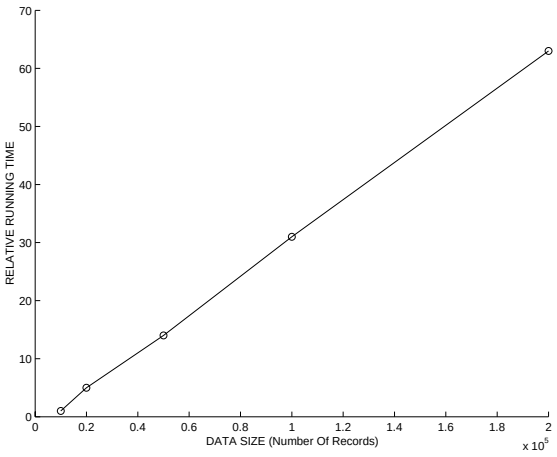


Figure 15: Efficiency of Reduction with Increasing Database Size (T15.I15.Dx)

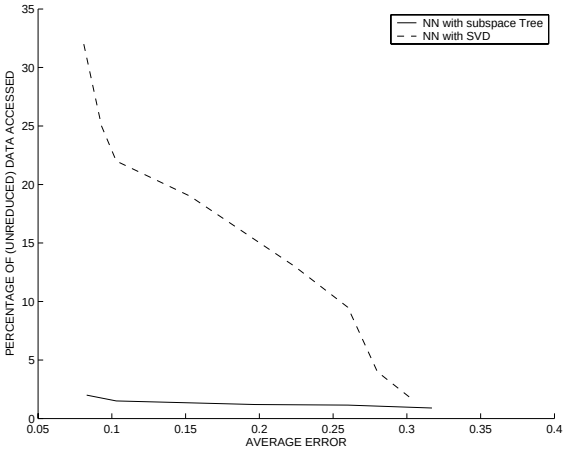


Figure 16: Pruning Performance for Approximate Nearest Neighbor Search (32-d color histograms)

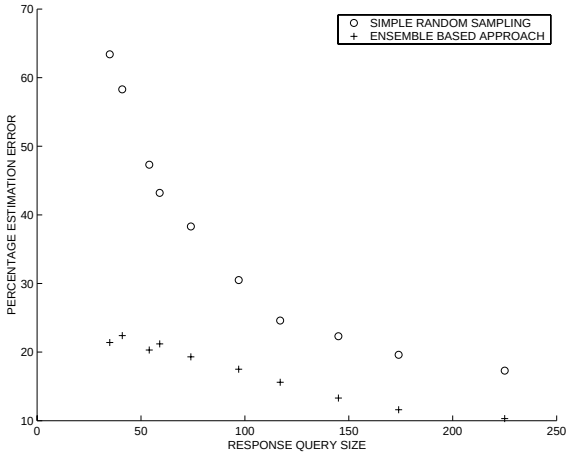


Figure 17: Effectiveness of Subspace Biased Sampling for Selectivity Estimation (32-d color histograms)

the hierarchical subspace sampling technique performed more effectively. This is because of the (almost) linear scalability of the subspace sampling technique with respect to data dimensionality, whereas the SVD method had worse than quadratic scalability with increasing dimensionality. We have also illustrated the scalability of the subspace sampling technique with increasing data set size in Figures 13 and 15 respectively. In this case, we used a 100-dimensional projection of the data sets T20.I20.D"x" and T15.I15.D"x" respectively. Here the value of x was varied in order to control the database size. It is clear that the subspace sampling technique scales almost linearly with database size. In fact in Figure 15, the subspace sampling approach scales slightly sublinearly with database size. The straightforward sampling approach is the key to the tremendous efficiency of the dimensionality reduction method.

5.3 Applications to Approximate Nearest Neighbor Search and Selectivity Estimation

We applied the subspace sampling method for approximate nearest neighbor search on the 32-d and 64-d color histogram data sets. A direct application of the branch and bound method [25] on the R-Tree structure leads to 100% of the data being accessed in each case. This is because of the high dimensionality of the problem which is outside the reach of normal index structures. However, a fairer comparison would be to combine SVD with the nearest neighbor indexing technique in order to present the results. We used the R-Tree structure to index a data set which was reduced using SVD, but with similar *average* error as the reduced data created by the subspace sampling

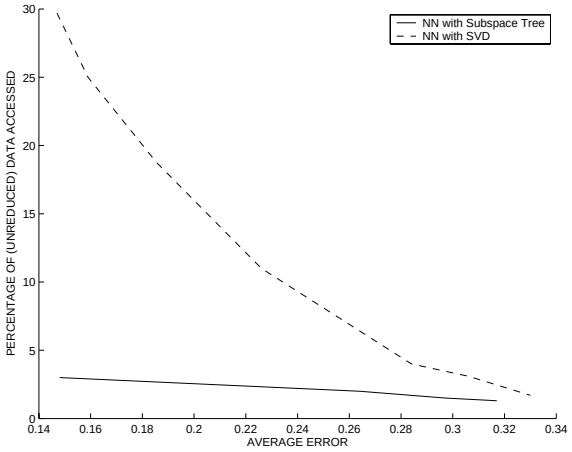


Figure 18: Pruning Performance for Approximate Nearest Neighbor Search (64-d color histograms)

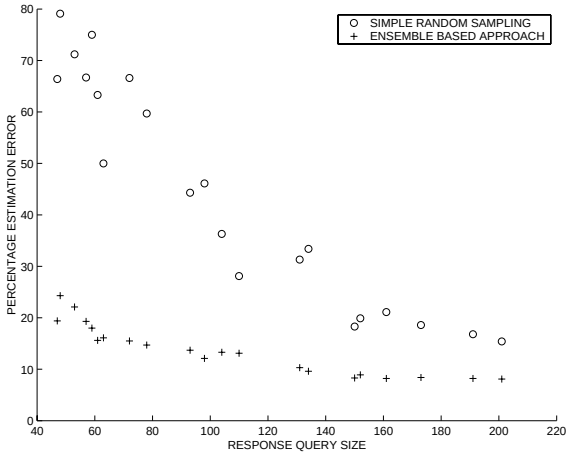


Figure 19: Effectiveness of Subspace Biased Sampling for Selectivity Estimation (64-d color histograms)

technique. In this case, the performance improved mainly because of the reduced data set size, whereas 100% of the index (on the *reduced* data) continued to be accessed in most cases. The results are illustrated in Figures 16 and 18. On the Y-axis of both figures, we have presented the results as a percentage of the *unreduced* data size, for consistency in comparison of the two methods. In the case of the 32-dimensional data set, the hierarchical subspace tree accessed between 0.9% and 2% of the data for all ranges tested. For the case of the 64-dimensional data set, the hierarchical subspace index tree accessed between 0.5% to 5% of the original data set size. The improvement of the subspace tree technique was *both* because of more effective pruning and the advantages of a reduced representation. Typically, between $60 - 90\%$ of the (reduced) data was pruned during the branch and bound search. On the other hand, in the case of the R-Tree structure, most of the data was accessed in almost all cases. Most of the reduction was obtained from the SVD compression. Unlike the R-Tree which used axis-parallel rectangles in characterizing the nodes of the partitions, the arbitrary hyperplanes of the subspace sampling technique provided tight bounds which helped in effective pruning during the nearest neighbor search procedure.

We also tested the ensemble-based approach for selectivity estimation on the 32-d and 64-d color histogram data sets. The queries were all range queries in which we intersected 10% on the ranges on two randomly picked dimensions. For the 64-dimensional case, simple random sampling is the most realistic alternative [13] for effective selectivity estimation. In Figures 17 and 19, we have illustrated the performance of the ensemble-based approach on the 32-d and 64-d color histogram data sets respectively. In each case, we used a representation factor of 3% and a maximum di-

mensionality of $q_{max} = 2$ for the ensemble-based estimator. Both approaches were implemented so that they required the same amount of storage space. In Figures 17 and 19, we have illustrated the estimation accuracy on the same queries for both methods. It is clear that the ensemble-based approach performs significantly more effectively than the simple random sampling procedure, especially for queries with small responses. Since the estimation of queries with small responses is the most inaccurate for most selectivity estimators, the overall robustness of the ensemble system was significantly better. The reason for this improvement was two-fold: (1) Since the histogram component of the ensemble was built in a lower dimensional space, it was more compact. Therefore, a greater amount of selectivity information could be stored in the same amount of space. (2) The primary reason for the effectiveness of the ensemble approach was its ability to decompose the data depending upon its natural degree of difficulty and use suitably optimized approaches for each of the portions. This generic approach can be leveraged to good use in a number of other high dimensional problems. We are currently exploring the use of this technique for high dimensional classification.

6 Conclusions and Summary

In this paper, we discussed the novel technique of hierarchical subspace sampling, a method for effective high dimensional data reduction. As indicated by the empirical results, the hierarchical subspace sampling technique is both effective and efficient and can achieve a clear advantage over widely used dimensionality reduction techniques such as SVD. The technique shows the behavior of improved reduction ratios with increasing database size. Since subspace sampling methods such as those discussed in [1] concentrate on (globally) sampling the dimensions rather than the points, they do not exhibit this behavior. In addition, the technique shows almost linear scalability of running time with increasing database size and dimensionality. This results in an approach which is significantly more efficient than SVD for data sets of higher dimensionality. The reason for the efficiency is rooted in its straightforward sampling approach, while retaining the power of finding local subspaces of appropriate dimensionality in which to represent the data. The local subspaces are found more effectively for larger database sizes, as a result of which the reduction ratio improves with increasing database size. The hierarchical subspace sampling method creates a reduced representation on which techniques such as nearest neighbor search and selectivity estimation can be applied directly without requiring an initial phase of reconstruction. This results in an extremely

efficient solution to these problems. Therefore, the locality specific approach to subspace sampling is not only valuable for data reduction, but also reveals important local subspace properties of the data, which can be leveraged in a wide variety of applications.

References

[1] D. Achlioptas. Database-friendly Random Projections. *ACM PODS Conference*, 2001.

[2] C. C. Aggarwal. Hierarchical Subspace Sampling: A Unified Framework for High Dimensional Data Reduction, Selectivity Estimation, and Nearest Neighbor Search. *ACM SIGMOD Conference*, 2002.

[3] C. C. Aggarwal, P. S. Yu. Finding Generalized Projected Clusters in High Dimensional Spaces. *ACM SIGMOD Conference*, 2000.

[4] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *VLDB Conference*, 1994. *Proceedings of the ACM SIGMOD Conference*, 2000.

[5] S. Babu, M. Garofalakis, R. Rastogi. SPARTAN: A Model-Based Semantic Compression for Massive Data Tables. *ACM SIGMOD Conference*, 2001.

[6] L. Brieman. Bagging Predictors. *Machine Learning*, 24:123-140, 1996.

[7] N. Beckman, H.-P. Kriegel, R. Schneider, B. Seeger. The R*-Tree: An Efficient and Robust Method for Points and Rectangles. *Proceedings of the ACM SIGMOD Conference.* 322–331, 1990.

[8] E. Bingham, H. Mannila. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. *ACM KDD Conference Proceedings*, 2001.

[9] K. Chan, W. Fu. Efficient Time Series Matching with Wavelets. *ICDE Conference*, 1999.

[10] K. Chakrabarti, S. Mehrotra. Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces. *VLDB Conference*, 2000.

[11] A. Deshpande, M. Garofalakis, R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. *ACM SIGMOD Conference*, 2001.

[12] C. Faloutsos, K.-I. Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. *ACM SIGMOD Conference*, 1995.

[13] D. Gunopulos, G. Kollios, V. Tsotras, C. Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. *SIGMOD Conference*, 2000.

[14] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the ACM SIGMOD Conference*, 47–57, 1984.

[15] P. Indyk, R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *ACM STOC Proceedings*, pages 604-613, 1998.

[16] H. V. Jagadish, J. Madar, R. Ng. Semantic Compression and Pattern Extraction with Fascicles. *VLDB Conference*, 1999.

[17] W. Johnson, J. Lindenstrauss. Extensions of Lipschitz mapping into a Hilbert space. *Conference in modern analysis and probability*, pages 189-206, American Math Society, 1984.

[18] I. T. Jolliffee. *Principal Component Analysis*, Springer-Verlag, New York, 1986.

[19] E. Keogh, S. Chu, M. Pazzini. Ensemble-index: A New Approach to Indexing Large Databases. *ACM SIGKDD Conference*, 2001.

[20] S. Berchtold, D. A. Keim, H.-P. Kriegel: The X-Tree: An Index Structure for High-Dimensional Data. *VLDB Conference*, 1996.

[21] K.-I. Lin, H. V. Jagadish, C. Faloutsos. The TV-tree: An Index Structure for High Dimensional Data. *VLDB Journal*, 3 (4), pages 517–542, 1992.

[22] D. A. Keim, M. Heczko. Wavelets and their Applications in Databases. *ICDE Conference*, 2001.

[23] E. J. Keogh, K. Chakrabarti, M. Pazzini, S. Mehrotra. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases, *KAIS*, 3(3): 263-286, 2000.

[24] E. J. Keogh, K. Chakrabarti, M. Pazzini, S. Mehrotra. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM SIGMOD Conference*, 2001.

[25] N. Roussopoulos, S. Kelley, F. Vincent. Nearest Neighbor Queries. *Proceedings of the ACM SIGMOD Conference*, pages 71–79, 1995.

[26] C. H. Papadimitriou, P. Raghavan, H. Tamaki, S. Vempala. Latent Semantic Indexing: A Probabilistic Analysis. *ACM PODS Conference*, 1998.

[27] V. Poosala, Y. Ioannidis. Selectivity Estimation without the Attribute Value Independence Assumption. *VLDB Conference*, 1997.

[28] V. Poosala, Y. Ioannidis, P. Haas, E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *ACM SIGMOD Conference*, 1996.

[29] K. V. Ravi Kanth, D. Agrawal, A. Singh. Dimensionality Reduction for Similarity Searching in Dynamic Databases. *SIGMOD Conference*, 1998.

[30] S. T. Roweis, L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, Vol 290, pp. 2323-2326, Dec 2000.

[31] J. B. Tenenbaum, V. Silva, J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, Vol 290, Dec 2000.

[32] D. Wu, D. Agrawal, A. Abbadi. A Comparison of DFT and DWT based Similarity Search in Time Series Databases, *Proceedings of the 9th International Conference on Information and Knowledge Management*, 2000.

[33] J. Ziv. A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transaction on Information Theory*, 23(3):337-343, 1977.