Clustering High-Dimensional Data Using an Efficient and Effective Data Space Reduction

March 3, 2005

Ratko Orlandic
Computer Science
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
ratko@cs.iit.edu

Ying Lai
Computer Science
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
laiying@iit.edu

Wai Gen Yee
Computer Science
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
emailyee@iit.edu

ABSTRACT
Clustering multi-dimensional data on storage in a way that maximizes the densities of storage clusters can greatly facilitate efficient and scalable retrieval of data. Clustering algorithms appropriate for this application must support efficient reduction of the data space that enables effective reduction of the search space. This paper introduces a new algorithm for clustering data in multi-dimensional feature spaces, called GARDEN$_{HD}$. Designed to support intelligent data storage, this algorithm employs an efficient data space reduction technique that scales both in terms of the number and dimensionality of data objects. It also produces a compact representation that can effectively capture the essence of the data. The algorithm is a hybrid of cell-based and density-based clustering. However, unlike typical clustering methods in its class, it applies a recursive partition of sparse regions in the space using a new space-partitioning strategy. The properties of this partitioning strategy greatly facilitate data space reduction. The experiments on synthetic and real data sets reveal that GARDEN$_{HD}$ and its data space reduction are effective, efficient, and scalable.

1. INTRODUCTION
Clustering is the process of grouping a set of data items based on the values of their attributes. Data items are generally interpreted as points in a multi-dimensional feature space. Items within the same cluster should be similar according to a similarity metric that is defined over the space (usually Euclidian distance). Conversely, items in different clusters should be dissimilar under the same metric.

Clustering multi-dimensional data has application in statistics, machine learning, pattern recognition, image processing, and data mining. In this paper, we consider the role of clustering multi-dimensional data on storage in a way that results in high-performance data access [2, 18], and introduce a new clustering method for this application.

We argue that, to support efficient and scalable retrieval of multi-dimensional data, database and data-warehousing systems should employ a clustered storage organization that maximizes the densities of clusters on storage. The clustering algorithm appropriate for this application must effectively isolate areas with points, reducing the amount of their internal empty space. In other words, it must support effective *data space reduction* that enables effective reduction of the search space. Furthermore, the clustering method must scale in terms of the number of points and number of dimensions. The clustering algorithm described in this paper, called GARDEN$_{HD}$ (*GAmma Region DENsity* clustering in *High Dimensionalities*), achieves both of these goals.

Related Work: Although various clustering methods have been proposed [9], they generally do not satisfy the above requirements. Those algorithms that require prior knowledge about the number of clusters, e.g. K-means [13] and CLARANS [15], do not facilitate effective data space reduction. BIRCH [22] also requires several sensitive input parameters, which are often hard to estimate ahead of time. Experimental studies [3, 20] show that the performance of CLARANS is close to quadratic in the number of points. While DBSCAN [3] generally outperforms CLARANS, it still does not perform well in high-dimensional situations [1]. WaveCluster [19] is an effective and efficient algorithm, but it can be applied to low-dimensional data only [6]. Similarly, STING [20] is an efficient algorithm, but it has no straightforward extension for high-dimensional data.

The clustering algorithms for high-dimensional spaces, e.g. DENCLUE [7], OptiGrid [8], CLIQUE [1], MAFIA [14], SUBCLU [11], and HyCeltyc [18], tend to be the hybrids of grid-based (cell-based) and density-based clustering. In these spaces, they employ some form of dimensionality reduction or derive the clusters in a subspace. Dimensionality reduction [4, 10] is a useful technique that works well on correlated data. Even then, it may cause a loss of clusters and the distortions of the densities and spatial properties of clusters, which does not facilitate data space reduction.

The incompatibility of most clustering algorithms with the concerns of data storage and retrieval is due to the fact that they are designed for data analysis. As a consequence, these algorithms usually eliminate “noise” in the data. When data needs to be stored, no data item can be eliminated—each should be assigned to a location in storage close to its spatial neighborhood. The best clustering algorithms for data analysis optimize their clustering criteria [5], usually at the expense of efficiency and scalability. Such algorithms are too heavy to be useful for storage organization. On the other hand, clustering methods for data storage can be useful in data analysis.

The need for new clustering methods designed to support effective storage and retrieval of multi-dimensional data has been emphasized in [2]. The authors propose a technique that finds correlated clusters and performs local dimensionality reduction on individual clusters, each of which is indexed separately. Our approach does not preclude dimensionality reduction, and it can even facilitate local dimensionality reduction. However, due to this focus on effective data space (rather than dimensionality) reduction, our work differs in the approach and the clustering technique.

Contributions: This paper conveys the importance of clustering for storage and retrieval of multi-dimensional data, and intrudes a data space reduction technique and a clustering algorithm for this application. Since the algorithm, called GARDEN$_{HD}$, satisfies the requirements stated earlier, it is appropriate for any intelligent system that supports advanced content-based retrieval, including data-warehousing systems as well as multimedia and scientific databases. However, experimental evidence indicates that the algorithm can also be effective as a general-purpose clustering method.

GARDEN$_{HD}$ is a hybrid of cell-based and density-based clustering [18, 19]. Due to its original technique for data space reduction, it represents a different approach to clustering, which is both effective and efficient. While GARDEN$_{HD}$ is appropriate in low-dimensional situations, the subscript of its name emphasizes the fact that it can operate directly in high-dimensional spaces without dimensionality reduction.

Effective space reduction implies the ability to represent compactly the essential properties of the data distribution. Even in high-dimensional spaces, the compact representation of data produced by GARDEN$_{HD}$ can preserve all essential properties of data, including the locations, shapes, and densities of clusters. As a result, many data mining tasks can be performed efficiently on this representation.

The efficiency and effectiveness of the proposed data space reduction technique is due to the application of a new space-partitioning strategy called γ, a variant of Γ partitioning [17], which has important advantages over traditional strategies. For example, when splitting each axis once, a grid-like partition [19, 20] of a D-dimensional region would produce 2^D sub-regions. Because of this exponential explosion, a high-dimensional region can be partitioned only along a small subset of dimensions. This can prevent differentiation of data along most dimensions and elimination of empty space that appears along these dimensions.

In contrast, γ can partition any given region along every side creating only $\mathrm{O}(D)$ sub-regions. Moreover, since the partition is performed at virtually negligible costs, both in terms of the memory and computational requirements, effective and efficient identification as well as separation of densely populated areas in the space becomes possible. A formal definition and an extensive treatment of the properties of γ partitioning appear in this paper for the first time.

Section 2 lays out the foundations of this work. Section 3 gives the definition of γ partitioning. Section 4 presents the space-reduction technique of GARDEN$_{HD}$. Section 5 gives the operational details of GARDEN$_{HD}$ and discusses its efficiency and scalability. Section 6 gives a comprehensive experimental evidence. Section 7 summarizes the paper and discusses broader applications of this technology.

2. MOTIVATION AND OBJECTIVES

While efficient clustering methods with effective data space reduction can have many applications, this section emphasizes the importance of clustering and data space reduction for storage and retrieval of multi-dimensional data. The discussion can be generalized for any kind of persistent storage. However, to simplify the discussion, we assume disk as storage media as well as data or index pages as the units of storage. For the same reason, we rely on simple formulas that reveal the most significant factors of retrieval performance, which we measure in terms of the number of accessed pages. As in the rest of the paper, we assume a normalized D-dimensional universe $[0,1]^D$.

Considering the general importance of clustering data on storage, let us assume for the moment an idealized system that, for any given query, accesses only those pages on secondary storage that contain useful data (data items that satisfy the query). Moreover, assume that N data items are divided into $M \ll N$ pages and that $K \ll N$ items satisfying the query are randomly distributed among the pages. In this case, the well-known Cardenas expression $\bar{A} = M(1-(1-1/M)^K)$ gives a good estimate of the number accessed pages for the given query [21]. Note that the number of pages with useful data is at most $min\{K, M\}$ and, when $K \ll M$, the above expression can be approximated by $\bar{A} \approx M(1-(1-K/M)) = K$.

Eliminating the assumption that data items are randomly distributed among the pages, the number of accessed pages can be estimated by a more general expression, valid for any $K, M \leq N$: $\bar{A} = K/\bar{I}$, where $\bar{I} \geq 1$ is the average number of items that satisfy the query in an accessed page. Since the goal of clustering data on storage is to increase the number of useful items in any page accessed by a typical query, the parameter $\bar{H} = \bar{I}/C \leq 1$, where C is the page capacity, is a good measure of the quality of clustering with respect to the given query. Obviously, when $C = 1$ or $\bar{H} = 1/C$, clustered storage organization has the same effect as the organization with randomly distributed data. However, for $C \gg 1$ and $\bar{H}$ as close to 1 as possible, the performance improvement can be significant (as much as C times fewer page accesses). For this to happen, the storage utilization must also be high.

Developing an appropriate clustering strategy, let us consider the problem of supporting multi-dimensional region queries, which restrict the ranges of values in one or more dimensions. We use the term *storage cluster* to denote a spatial region containing points in a storage unit, i.e. a page.

In this context, increasing $\bar{H}$ means increasing the probability that each storage cluster with useful data is completely covered by the query. However, since clustering must benefit not one but many different queries, the storage organization must be such that, for every storage cluster S and any query Q, it decreases the probability $P(S \cap Q)$

that S overlaps Q (which would trigger access to the corresponding page), while increasing the probability $P(S \subseteq Q)$ that it is covered by Q. Assuming that S is represented by its minimum bounding hyper-rectangle, and that all possible positions of S are equally likely, one can show that: $P(S \cap Q) = \prod_{i=1}^{D} min\left\{Q_i + S_i, 1\right\}$ and $P(S \subseteq Q) = \prod_{i=1}^{D} max\left\{\frac{Q_i - S_i}{1 - S_i}, 0\right\}$, where S_i and Q_i are the extensions (lengths) of S and Q, respectively, in an axis i.

Since the extensions of any given query are fixed, the way to reduce $P(S \cap Q)$ and increase $P(S \subseteq Q)$ for an arbitrary query Q is to reduce the lengths S_i of S along all dimensions i that are restricted by Q (i.e., all i for which $Q_i < 1$). With this and the earlier observation about the storage utilization, we can now formulate an important principle: multi-dimensional data must be grouped on storage in a way that minimizes the extensions of storage clusters along all relevant dimensions and achieves high storage utilization.

The term "relevant dimensions" refers to the fact that queries may have "affinity" for certain dimensions, consistently leaving other dimensions unrestricted. However, if this is not known *a priori*, or if the query pattern may change over time, the clustering scheme must treat all dimensions as equally important.

Assuming a multi-dimensional space defined by relevant dimensions, the stated principle implies that the storage organization must maximize the densities of storage clusters both by increasing the number of points in the clusters and by reducing their volumes. To increase the densities of storage clusters, the organization must reduce their internal empty space. For the best results, the database system should employ a genuine clustering algorithm that can efficiently isolate densely populated areas in the space, effectively reducing the amount of empty space.

We refer to the process of detecting dense areas (*dense cells*) in the space with minimum amounts of empty space as *data space reduction*. In this context, *data clustering* is a process of detecting the largest areas with this property, called *data clusters*. Depending on the environment, the stated design principle can be achieved either by clustering or data space reduction only. A capability to perform either or both can benefit any environment.

The above discussion assumes region queries. However, data clustering with effective data space reduction can facilitate various kinds of retrieval, including similarity searching, by enabling: a) a close-to-optimal assignment of data to pages; and b) a significant reduction of the search space even before the retrieval process hits persistent storage. For effective data space reduction, the clustering method should operate directly in the externally defined space without dimensionality reduction, and it should not be governed by any expectation about the number of clusters. To be useful for storage organization, it must also be very efficient. This set of requirements motivates the design of GARDEN$_{HD}$.

3. SPACE PARTITIONING STRATEGY

In this section, we present a new variant of the Γ partitioning strategy [17] used in GARDEN$_{HD}$. Like Γ, this strategy eliminates the exponential explosion as dimensionality grows that accompanies grid partitioning strategies, which was discussed in Section 1. In some ways, this strategy is a specialization and, in others, generalization of Γ. In order to denote the fact that this is a different partitioning strategy descending from Γ, we refer to it as γ *partitioning*.

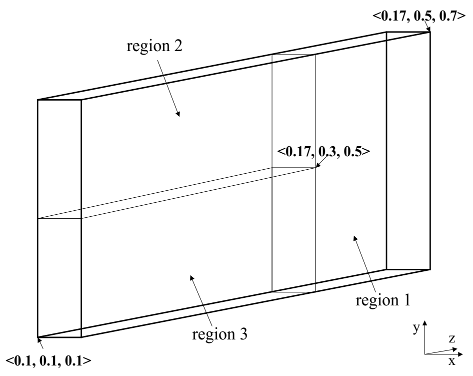

Figure 1: A γ partition of a 3D base region.

To formally define γ partitioning, we assume a D-dimensional feature space U and use: $\vec{p}$ and p_i to denote a vector (point) in U and its i-th coordinate, respectively; $R = \langle \vec{l}, \vec{h} \rangle$ to denote a hyper-rectangle in U defined by its low ($\vec{l}$) and high ($\vec{h}$) endpoint; $[i, j]$ to denote a range of values between i and j inclusively; and "$\leftarrow$" to denote an assignment.

Definition 1: *Let $\vec{d}$ define an order of dimensions. Let $k \in [1, D]$ be an integer; $B = \langle \vec{l}^{(B)}, \vec{h}^{(B)} \rangle$ a hyper-rectangle in U, s.t. $\forall i \in [1, k]$, $l_{d_i}^{(B)} < h_{d_i}^{(B)}$; and $\vec{g}$ a vector in B, s.t. $\forall i \in [1, k]$, $l_{d_i}^{(B)} < g_{d_i} < h_{d_i}^{(B)}$, $\forall i \in [k+1, D]$, if any, $g_{d_i} = h_{d_i}^{(B)}$. Then, $\gamma = \langle B, \vec{d}, k, \vec{g} \rangle$ is a well-formed γ partition of B into $k+1$ non-overlapping hyper-rectangles $G^{(i)} = \langle \vec{l}^{(i)}, \vec{h}^{(i)} \rangle$ provided $\vec{l}^{(i)}$ and $\vec{h}^{(i)}$ can be derived as follows: $\forall i \in [1, k]$, $\vec{l}^{(i)} \leftarrow \vec{l}^{(B)} \wedge l_{d_i}^{(i)} \leftarrow g_{d_i}$; $\vec{l}^{(k+1)} \leftarrow \vec{l}^{(B)}$; $\vec{h}^{(1)} \leftarrow \vec{h}^{(B)}$; and $\forall i \in [2, k+1]$, $\vec{h}^{(i)} \leftarrow \vec{h}^{(i-1)} \wedge h_{d_{i-1}}^{(i)} \leftarrow g_{d_{i-1}}$.*

Informally, the definition says that each region $G^{(i)}$ in a γ partition of a hyper-rectangle B is derived iteratively, in the order defined by $\vec{d}$, by setting at most one coordinate in the low endpoint of B and the high endpoint of the previous region $G^{(i-1)}$, if any, to the corresponding coordinate of a given vector $\vec{g} \in B$. Each coordinate d_i of $\vec{g}$, where $i \leq k$, gives the position of a hyper-plane that separates $G^{(i)}$ from the space in which the regions $G^{(j>i)}$ lie. In the rest of the paper, we refer to B as *base region*, $\vec{d}$ as *vector of dimensions*, $\vec{g}$ as *generating vector*, and each $G^{(i)}$ as a γ *region*. We refer to k as the *number of split dimensions* because B is split once along $k \leq D$ different sides.

Figure 1 shows a γ partition of a 3D base region whose one side (dimension x) is much shorter than the other two (y and z). The example assumes that $\vec{d} = \langle z, y, x \rangle$, and that only the first $k = 2$ dimensions are partitioned. This partition has three γ regions. As in all γ partitions, the high endpoint of the last region represents the generating vector $\vec{g}$.

The Γ strategy [17] is more restrictive than γ in that it assumes: $B = U$, $\vec{d}$ is fixed, and $k = D$. However, Γ allows more than one generating vector (i.e., it can split each side of B more than once), each of which must satisfy a less restrictive "nesting" condition than $\vec{g}$ in Definition 1. Using a single generating vector ensures that the number of γ regions is at most $D+1$, enabling a tightly bound space and time complexity.

Explicit in Definition 1 is an important property of γ partitions: a γ partition is fully defined by $\langle B, \vec{d}, k, \vec{g} \rangle$. Thus, to define a γ partition, one must reserve memory for only O(1) D-dimensional vectors, including the low and high endpoints of the base region B. We will later see that, for a given B, a γ partition can be derived within the computational complexity equivalent to O(1) point comparisons.

Since the data-space reduction technique must detect the empty space that separates any two disjoint dense cells, which can appear along any or multiple dimensions, this ability of γ to efficiently split a region along any number of dimensions is very important for accurate and efficient data space reduction. Equally important is the following property of γ partitions:

Lemma 1: *Given a well-formed partition* $\gamma = \langle B, \vec{d}, k, \vec{g} \rangle$ *and an arbitrary point* $\vec{p} \in B$*:* a) $\forall i \in [1, k]$, $\vec{p} \in G^{(i)}$ *iff* $p_{d_i} \geq g_{d_i} \wedge \forall j \in [1, i-1]$, *if any,* $p_{d_j} < g_{d_j}$*; and* b) $\vec{p} \in G^{(k+1)}$ *iff* $\forall j \in [1, k]$, $p_{d_j} < g_{d_i}$.

Informally, Lemma 1 states that, for any given point that lies within the base region, one can identify the γ region in which the point belongs just by comparing the coordinates of the point against the corresponding coordinates of the generating vector $\vec{g}$ in the order determined by $\vec{d}$. Thus, the time required to locate the γ region where the point belongs is O(1) comparisons of D-dimensional points. The proof of Lemma 1 follows straightforwardly from Definition 1 after observing that each γ region $G^{(i)}$, $i \leq k$, lies above the hyper-plane that separates it form the space in which the subsequent regions $G^{(j>i)}$ lie.

A γ partition induces non-overlapping regions that cover the extent of the base region B. However, the base region may have large amount of empty space. In order to isolate areas of B containing points, a *live region* is computed within each of its γ regions. The live region is the minimum bounding hyper-rectangle containing all points in the γ region. With Lemma 1, the process of data space reduction has a basis for efficient computation of live regions. This important lemma will also serve as the basis for efficient agglomeration of dense cells detected during data space reduction.

4. DATA SPACE REDUCTION

This section introduces the data-space reduction technique of GARDEN$_{HD}$. This recursive density-based technique uses γ partitioning, an efficient method to isolate areas with points, and several heuristics designed to deal with certain

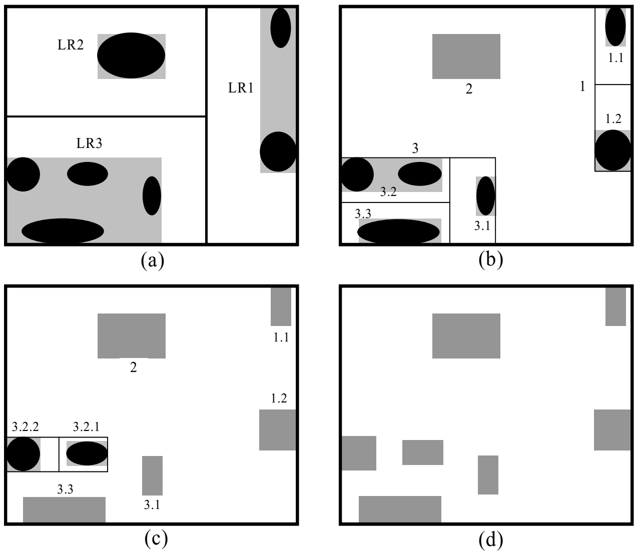

Figure 2: Example of recursive space reduction.

"false dense cells" that can bridge distant clusters. The result is a vastly reduced data space represented as a set of dense cells. The algorithm of the technique appears in Section 5, as part of the algorithm of GARDEN$_{HD}$. Beginning with a brief description of the technique, this section presents the details of its constituent mechanisms.

4.1 General Operation

Given a density threshold δ as the only input, the process of data space reduction starts by scanning the data once in order to compute the live region LR_U enclosing all points in the universe. This becomes the initial base region. The technique proceeds by partitioning LR_U into multiple γ regions. Within each γ region, its live region is identified. Sparse live regions are recursively partitioned into smaller regions using γ partitioning and live-region identification, until each of the smaller live regions has a density of at least δ, becoming a dense cell. The process produces a compact "signature" of data consisting of the identified dense cells, represented by their low and high endpoints.

The process is illustrated in Figure 2. Figure 2a, in which the dark ovals represent areas populated by points, shows the γ partition of the initial base region as well as the live regions within the resulting γ regions. In Figures 2b and 2c, sparse live regions $LR1$ and $LR3$ are partitioned further until all their enclosed dense cells are identified. Whenever a high-density live region is detected (e.g., $LR2$ in Figure 2a), it is included into the set of dense cells. The resultant dense cells of this example are shown in Figure 2d.

4.2 Region Partitioning

The γ partition of a base region is performed by the procedure SplitBaseRegion of Figure 3. Given a base region B, a vector of dimensions $\vec{d}$ (called DV in the procedure) and the number of split dimensions $k \leq D$, the procedure

```
PROCEDURE SplitBaseRegion
 INPUT
   $B$;          // base region
   $DV$;         // vector of dimensions, i.e. $\vec{d}$
   $k$;          // number of split dimensions
 OUTPUT
   $GV$;         // generating vector, i.e. $\vec{g}$
 BEGIN
   $R \leftarrow B$.Region;              // temporary region
   $scale \leftarrow 1 \,/\, (1 + k)$;   // initial scale
   for $i = 1$ to $k$
     $j \leftarrow DV[i]$;               // axis to be split
     $R[\text{high},j] \leftarrow R[\text{high},j]$ - $(R[\text{high},j]$ - $R[\text{low},j]) \cdot scale$;
     $GV[i] \leftarrow R[\text{high},j]$;
     $scale \leftarrow scale \,/\, (1$ - $scale)$;
   end for
 END
```

Figure 3: Procedure that partitions a base region into γ regions of equal volume.

completes the γ partition by deriving a generating vector $\vec{g}$ (called GV in the procedure) that produces $k+1$ γ regions of equal volume.

For simplicity of density computations, GARDEN$_{HD}$ assumes a normalized universe $[0,1]^D$. However, SplitBaseRegion has no such restriction. The base region can have any hyper-rectangular shape and, in principle, the dimensions can be of any numeric type. The origin of the base region can lie anywhere in the space. Observe that the γ partition of Figure 1 was derived using this procedure.

In order to compute each coordinate of the generating vector, SplitBaseRegion examines two coordinates of the temporary region R and performs a small number of single-value assignments. Thus, the computational complexity of SplitBaseRegion is equivalent to O(1) point comparisons.

A γ partition can be derived using the statistics about the data distribution in the given base region. However, this can lead to much higher computational complexity or many more levels of recursion. In Setion 6.1, we will investigate whether the adopted partitioning into γ regions of equal volume is adequate for the purposes of effective data space reduction.

4.3 Identification of Live Regions

As noted earlier, for each γ region, its live region is computed. In the context of data space reduction, these live regions serve a dual purpose: to eliminate large empty areas that characterize typical (especially high-dimensional) data spaces, greatly contributing to the effectiveness of the space reduction; and to reduce the volume of space that needs to be inspected, increasing the efficiency of the reduction.

Figure 4 gives the procedure FindLiveRegions, which computes the live regions of the γ regions produced by partitioning a base region B. Drawing on the property of γ partitions stated in Lemma 1, this procedure performs a very efficient identification of live regions. For a given point that lies within the base region, it locates the γ region $G^{(i)}$ in which the point belongs just by comparing the point against the generating vector $\vec{g}$ (i.e., GV) one coordinate at a time.

```
PROCEDURE FindLiveRegions
 INPUT
   $B$, $DV$, $k$, $GV$;      // same as in SplitBaseRegion
 OUTPUT
   $LRL$;                     // list of live regions
 BEGIN
   $LRL$ = InitializeLiveRegions $(k+1)$;
   for each $point \in B$
     $i \leftarrow 1$;
     $j \leftarrow DV[i]$;              // examine the first coordinate
     while $i \leq k \wedge point[j] < GV[j]$
       $i \leftarrow i + 1$;
       $j \leftarrow DV[i$ mod $k]$; // examine the next coordinate
     end while
     UpdateLiveRegion $(LRL, i, point)$;
   end for
   UnlinkEmptyLiveRegions $(LRL)$;
 END
```

Figure 4: Procedure that computes live regions.

For the best performance, this procedure must directly access all data points in the base region B. For the time being, we assume a simple array of points internally organized as a set of linked lists, each of which is anchored in the memory block of a live region. Each element of this structure contains a data point and a pointer to the next data point in the live region.

After FindLiveRegions determines the γ region for the given point p, it updates the corresponding live region LR and links p to the points of LR. If LR is later partitioned, its points can be accessed directly through an anchor pointer in the memory block of LR.

The time required to determine the γ region where a data point belongs is O(1) comparisons of D-dimensional points, and so is the time to update the corresponding live region. Since only the points in the base region B are accessed, the complexity of FindLiveRegions is O(N') point comparisons, where N' is the number of points in B.

4.4 Elimination of False Dense Cells

As noted in Section 1, traditional grid partitioning strategies can split a high-dimensional region only along a subset of its sides. As a result, a combination of grid partitioning and live-region identification would lead to the creation of many sparse and elongated live regions. Even worse, such regions could appear dense even if each has just few distant points.

For example, if the length of one side of a live region LR is less than $2/\delta$, the density of LR in the given space is greater than δ regardless how long the other sides are or how many points fall in LR. These *false dense cells* (elongated sparse live regions that appear dense) may contain large amounts of empty space, and they may erroneously “bridge” distant clusters, creating an illusion of a single cluster.

Since γ partitioning can perform efficient partitioning of a region along all its sides, it has a natural mechanism that reduces the magnitude of this problem. Yet even with γ, these false dense cells may still appear. To deal with the problem, our space-reduction technique applies a set of simple heuristics incorporated in two procedures, called ConstructDV and SparsityTest. The former tries to reduce the likelihood of false cells; the latter tries to detect them, in which case they are (just as other sparse regions) split.

Before partitioning a sparse region B, the ConstructDV procedure determines $k \leq D$ sides of B that must be partitioned, and returns a vector of dimensions $\vec{d}$ in which these sides are ordered first. The lengths of these sides must be greater than a computed value λ (see below) and at least half the length of its longest side. Since the procedure requires only two passes over the coordinates of B, its complexity is O(1) point comparisons. With the given $\vec{d}$ and k, the procedure SplitBaseRegion partitions B only along these k sides in the order given by $\vec{d}$ (this is why the regions $LR1$ and $LR3.2$ in Figures 2b and 2c, respectively, are split into only two regions of equal size.) This measure improves the "squareness" of not only the resulting γ regions, but also their live regions, making the false dense cells less likely to appear.

The boolean function SparsityTest determines if a live region is sparse or not. In order to facilitate this function, when the program begins, GARDEN$_{HD}$ computes an array Δ of density thresholds for all dimensionalities $i \leq D$ based on the given density threshold δ. Using δ, it computes a value λ for which a region of size λ^D with just two points is dense. From $2/\lambda^D = \delta$, one obtains $\lambda = (2/\delta)^{1/D}$. Then, $\Delta_i \leftarrow \delta$, if $i = D$, and $\Delta_i \leftarrow 2/\lambda^i$, if $i < D$.

SparsityTest computes the density of a live region LR in its *effective dimensionality* $D' \leq D$, ignoring all sides of LR that are shorter than $2/\delta$. It also computes the density $density_1(LR)$ in the LR's longest side alone. If either $density_1(LR) < \Delta_1$ or $density_{D'}(LR) < \Delta_{D'}$, then LR is considered sparse and, thereby, split. The test against Δ_1 is designed to avoid narrow dense cells with very few points.

5. GARDEN-HD CLUSTERING

Conceptually, the recursive space reduction creates a tree, where the leaves represent dense cells and the interior nodes represent instances of γ partitioning. As the recursion folds back, GARDEN$_{HD}$ agglomerates adjacent dense cells, forming the final set of clusters. Like the space-reduction technique, GARDEN$_{HD}$ requires only one input parameter, i.e. the user-defined density threshold δ. The result is a *cluster representation* of data, which represents each cluster by the list of its internal dense cells.

Agglomeration can be approached in a variety of different ways. In general terms, the choice is between density-connected and adjacency-connected agglomeration. The latter approach cannot guarantee that the derived clusters have density above the selected δ. However, even without this, one can effectively achieve all objectives stated in Section 2 by considering dense cells within sparse clusters. At the same time, the adjacency-connected approach enables a more efficient agglomeration that operates solely on dense cells identified by the data space reduction. Since the unions of adjacent cells can be arbitrarily shaped, it also enables a more general clustering solution. For these reasons, we adopt the adjacency-connected approach.

```
PROCEDURE SplitAndMerge
 INPUT
   $B$;                // base region; initially $LR_U$
   $CL$;               // list of clusters; initially empty
 BEGIN
   $\langle DV, k \rangle \leftarrow$ ConstructDV ($B$);
         // determine DV ($\vec{d}$) and no. of split dimensions $k$
   $GV \leftarrow$ SplitBaseRegion ($B$, $DV$, $k$);
         // obtain the $\gamma$ partition of the base region
   $LRL \leftarrow$ FindLiveRegions ($B$, $DV$, $k$, $GV$);
         // find live regions; return the list of live regions
   for each $LR \in LRL$
       if SparsityTest ($LR$)
            // low-density region that must be split further
           SplitAndMerge ($LR$, $CL$);
       else // a dense cell identified
           AddDenseCell ($LR$, $LR$);
            // the dense-cell list of LR is LR itself
           AddNewCluster ($CL$, $LR$);
            // temporarily, treat LR as a new cluster
       end if
   end for     // END OF REDUCTION
   $LR \leftarrow$ GetFirstLR ($LRL$);
   while $LR \neq null$
     AttachDenseCellsToB ($B$, $LR$);
     if ProximityTestForLR ($LR$)
       $LR1 \leftarrow$ GetNextLR ($LRL$, $LR$);
       while $LR1 \neq null$
         if ProximityTestForLR1 ($LR$, $LR1$)
           for each $DC \in LR$
             if ProximityTestForDC ($DC$)
               for each $DC1 \in LR1$
                 $flag \leftarrow$ ProximityTestForDC1 ($DC$, $DC1$);
                 if  $flag \wedge$ Connected ($DC$, $DC1$)
                   MergeClusters ($CL$, $DC$, $DC1$);
                 end if
               end for
             end if
           end for
         end if
         $LR1 \leftarrow$ GetNextLR ($LRL$, $LR1$);
       end while
     end if
     $LR \leftarrow$ GetNextLR ($LRL$, $LR$);
   end while  // END OF AGGLOMERATION
 END
```

Figure 5: The core procedure of GARDEN$_{HD}$.

The two processes of GARDEN$_{HD}$ (*data space reduction* and *agglomeration*) are performed by the recursive procedure SplitAndMerge of Figure 5. The first invocation of this procedure takes the initial base region LR_U as input. The procedure has two parts, one for space reduction and the other for agglomeration. Note that, with only the first part, the procedure would support a stand-alone process of data space reduction. Since the steps of the first part are explained in Section 4, we focus now on the second part.

5.1 Efficient Agglomeration

In GARDEN$_{HD}$, two cells are merged when their distance along every dimension is below the minimum distance μ, which is derived from the density threshold δ. More precisely, μ is set to λ, where $\lambda = (2/\delta)^{1/D}$ was introduced in Section 4.4. By linking the minimum distance to the density threshold, the algorithm provides a single mechanism to regulate the space reduction and separation of data.

```
$\mu$;            // distance threshold
$DV$;             // vector of dimensions, i.e. $\vec{d}$
$GV$;             // generating vector, i.e. $\vec{g}$
$LR$, $LR1$;      // live regions of $\gamma$ regions $G^{(i)}$ and $G^{(j>i)}$
$DC$, $DC1$;      // dense cells in $LR$ and $LR1$, respectively
$x = DV[i]$;

ProximityTestForLR:
  $GV[x] \geq LR$.Region[$low$,$x$] - $\mu$

ProximityTestForLR1:
  $LR1$.Region[$high$,$x$] $\geq LR$.Region[$low$,$x$] - $\mu$

ProximityTestForDC:
  $GV[x] \geq DC$.Region[$low$,$x$] - $\mu$

ProximityTestForDC1:
  $DC1$.Cluster $\neq DC$.Cluster $\wedge$
  $DC1$.Region[$high$,$x$] $\geq DC$.Region[$low$,$x$] - $\mu$
```

Figure 6: The proximity tests in SplitAndMerge.

When the procedure SplitAndMerge begins agglomeration, the dense cells of each live region LR in the list LRL are identified and linked to the memory block of LR. Note that these live regions appear in the list LRL in the order of their corresponding γ regions. For each LR in LRL, its dense cells are immediately attached to the dense-cells list of the enclosing base region B and tested for connectivity with the dense cells of subsequent live regions in LRL.

Since the complexity of agglomeration is determined by the number of times the connectivity test Connected is invoked, the "proximity" tests in the procedure SplitAndMerge are used to eliminate unnecessary invocations of Connected based on one or two comparisons of numeric values. These tests also rely on the properties of γ partitioning.

For example, let LR be a live region of the γ region $G^{(i)}$ created by partitioning the base region B using the vector of dimensions $\vec{d}$. Based on Lemma 1, the coordinate d_i of each point $\vec{p}$ in a γ region $G^{(j>i)}$, if any, is less than g_{d_i}, which defines the position of the separating plane for the γ region $G^{(i)}$. Then, if LR is more than the distance threshold μ apart from the separating plane of $G^{(i)}$, the live regions of all subsequent γ regions must be more than the distance μ away from LR along the dimension d_i. If so, LR is enclosed by $G^{(i)}$ in such a way that no dense cell of any subsequent live region can be merged with any dense cell in LR.

Figure 6 gives the details of the proximity tests, each of which returns *true* if the corresponding condition is satisfied. In ProximityTestForDC1, DC.Cluster represents a parameter in the memory block of a dense cell DC indicating the cluster to which DC belongs at the given moment.

5.2 Efficiency of GARDEN-HD

Even though GARDEN$_{HD}$ does not restrict the depth of recursion, for a relatively large number of points N, virtually any data distribution, and almost any δ, the depth of the recursion is $O(\log N)$. For a uniform distribution, γ partitioning of a base region produces a balanced distribution of points among the resulting γ regions, ensuring $O(\log N)$ levels of recursion regardless of δ. For skewed data, the fact that γ partitioning is performed on live regions, and that it is multi-way partitioning, ensures that the volumes of live regions at some level $O(\log N)$ are so small that they would be dense even when the selected δ is extremely high.

The cost of the data space reduction is dictated by the procedure FindLiveRegions (all other procedures of the reduction require $O(1)$ point comparisons). Referring back to Section 4.3, one can see that, to derive all live regions at any level of recursion, at most N points will be examined, no point will be examined twice, and each point will be processed in time equivalent to $O(1)$ point comparisons. Thus, since $O(\log N)$ is virtually guaranteed depth of the recursion, $O(N \log N)$ point comparisons is virtually guaranteed cost of the recursive data space reduction.

On the other hand, the agglomeration is more sensitive to the selected value of δ, and the combination of very high δ and D can lead to $O(N^2)$ time.[1] However, assuming that $N >> D$, δ can be selected so that the number L of the resulting dense cells is at most $O(N/D)$ and that the minimum distance $\mu = (2/\delta)^{1/D}$ is a small value. Even without trials on the data, but with some prior thinking how to select δ, both of these conditions would typically be satisfied.

When this is the case, the neighborhood of any dense cell is small. For any two live regions LR and $LR1$ in a γ partition and a dense cells DC in LR, typically just a few cells in $LR1$ closest to DC will require a full test for connectivity with DC. Due to the "proximity" tests in the procedure SplitAndMerge, other cells in $LR1$ are quickly eliminated from inspection, either because they are immediately known to lie too far from DC or because they already belong to the same cluster as DC. Since there are at most D live regions that need to be examined at any level of recursion, $O(D)$ gives a realistic estimate of the number of neighboring cells examined for connectivity with DC, leading to just $O(D)$ point comparisons. Since at any level of recursion at most L dense cells are examined and the recursion is $O(\log N)$ levels deep, the cost of agglomeration is $O(DL \log N)$. When L is at most $O(N/D)$, this cost is $O(N \log N)$.

Thus, with a simple data structure and a bounded parameter δ, the total cost of GARDEN$_{HD}$ clustering is just $O(N \log N)$ comparisons of D-dimensional points. This estimate implies linear performance deterioration as dimensionality grows.

5.3 Large Sets and Parallelism

The data structure assumed so far (an array of points) is adequate for relatively small data sets. For large sets, it could lead to virtual-memory thrashing. Since the data points examined at any time may be scattered across the array, accessing a point could trigger an IO operation to fetch the corresponding block of data from the swap area on disk.

For this reason, our implementation of GARDEN$_{HD}$ keeps data points in a structure called the γ *tree*. In the structure, the points are organized into sub-lists according to their enclosing live regions at the first or second level of recursion.

[1]Turning the minimum distance μ into an input parameter would reduce the sensitivity on δ at the cost of having an additional parameter to be estimated in advance.

Each sub-list is stored in a contiguous block of memory. Due to the recursive nature of space partitioning, the sub-lists are accessed one at a time. Since each sub-list typically occupies a relatively small block of memory, the likelihood that an access to a data item would trigger an IO operation is reduced.

For very large sets, data points should be stored in a B^+-tree on disk and indexed by the unique numbers of their enclosing γ regions at some level of recursion (e.g., 2), which would depend on the data volume and dimensionality. Pre-partitioning the space (e.g., on a sample of the data) up to this level, would guarantee one-pass construction of the B^+-tree. Asynchronously retrieving the data from the B^+-tree one sub-list at a time would significantly reduce the memory requirements of the γ tree, which would have to reserve enough memory for only the few largest sub-lists. A similar arrangement would facilitate parallel processing of large data sets.

6. EXPERIMENTAL RESULTS

A comprehensive set of experiments was performed to observe the effectiveness and efficiency of GARDEN$_{HD}$ as well as the effectiveness of its data space reduction. The experiments were performed on a PC with a 3.2GHz CPU, 1GB memory, and 1MB CPU cache. Most of them were performed on:
1) "center-corners" – synthetic data of varying dimensionality (up to 100) and varying number of points (between 100,000 and 500,000); in high-dimensional spaces, the pre-computed clusters were placed in the center and 10 corners of the space (origin, far corner, and 8 randomly selected corners);
2) "covtype" – a set of real data with 54 dimensions (the 55th dimension records the class of objects) and 581,012 points; obtained from the UCI Machine Learning Repository (www.ics.uci.edu/~mlearn/MLRepository.html);
3) "animals" – a set of synthetic data with 72 dimensions (the 73rd dimension records the class of objects) and 500,000 points; produced by the "animals.c" program obtained from the UCI ML Repository.

6.1 Effectiveness of Data Space Reduction

The first two rows of Figure 7 give the visual representations, generated in MATLAB, of four synthetic distributions (each with 100,000 points) in a 3D space. From the point of view of space reduction, each of these distributions represents an interesting and non-trivial scenario. These pictures also display the front edges of dense cells detected by our space-reduction technique. The left pictures in the last two rows show two normalized 3D projections of a set of real data with 11 attributes and 8,284 points found in a repository of environmental data. The corresponding pictures on the right show the areas covered by dense cells detected by the recursive space reduction.

From the figure, one can visually observe that the space reduction is very effective. Note also that the set in the fourth row consists of correlated data and that the generated dense cells preserve this correlation. As in this case, the data space reduction technique can effectively capture the essence of any data set in the resulting representation. Therefore,

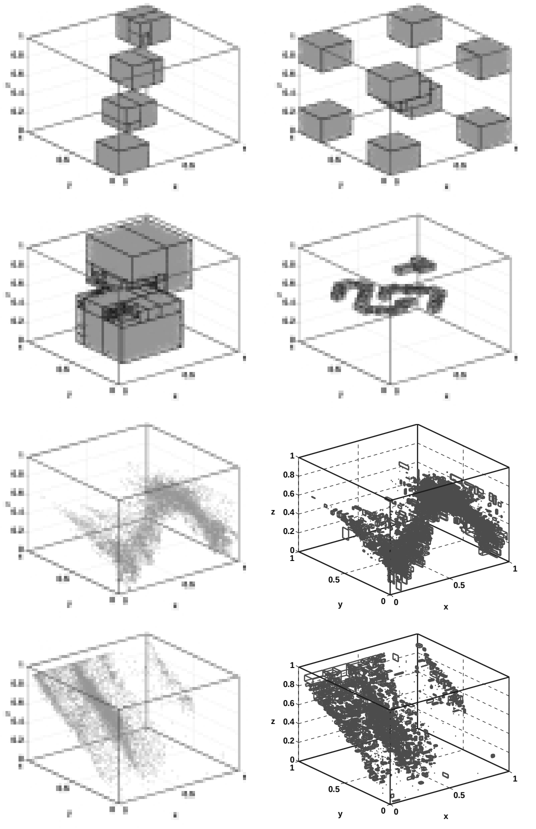

Figure 7: Visually observable effectiveness of the space reduction for synthetic and real 3D data.

many data mining tasks can be performed both effectively and efficiently on this representation alone.

The effectiveness of the space reduction in higher dimensionalities was monitored using the *adjusted volumetric measure* $\mathcal{M}_Q = \sum_i^L {V_i'}^{D/D'}$, where L is the number of dense cells and V_i' is the volume of the i-th cell in its effective dimensionality D' (not counting the sides that have extensions 0). Since dense cells whose computed volume would otherwise be 0 are treated as D-dimensional cells with a non-0 volume, $\mathcal{M}_Q$ can produce values greater than 1. However, we adopted this metric because it enables effective tracking of space reduction in progress and provides a good idea how much empty space is eliminated. Without the exponent D/D', $\mathcal{M}_Q$ could heavily distort the sense of the latter. Since data space reduction is an implicit data-reduction mechanism, we also monitor the *data-reduction ratio* $\mathcal{R}_Q = 2L/N$. Assuming no compression, $\mathcal{R}_Q$ indicates how much storage all dense cells occupy relative to the size of the data file.

Table 1, where the notation "E-y" stands for 10^{-y}, gives the results obtained on the "center-corners", "covtype", and "animals" sets. The third column gives the natural logarithm of the parameter δ (the choice of density thresholds is explained in the next sub-section). The fourth column (Q)

Table 1: Space reduction on different data sets.

Data Set	D	$\ln(\delta)$	Q	L	$\mathcal{R}_Q$	$\mathcal{M}_Q$
ccorners	20	50.7	8	368	0.0073	8.9E-18
ccorners	60	133.9	9	489	0.0098	6.0E-54
ccorners	100	217.1	8	755	0.0151	3.8E-90
covtype	54	80	8	1170	0.0040	6.1E-18
covtype	54	100	9	3143	0.0108	2.4E-24
animals	72	160	6	7402	0.0296	6.2E-65
animals	72	180	7	9210	0.0368	6.8E-74

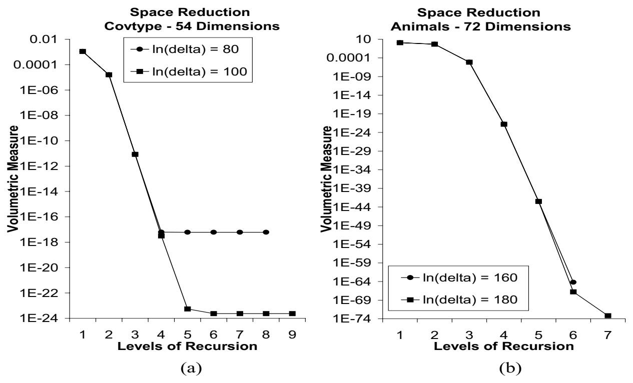

Figure 8: Space reduction on (a) "covtype" and (b) "animals" at each level of recursion (log scale).

gives the depth of the recursion. On the "center-corners" sets, each with 100,000 points, we varied the dimensionality D. On the "covtype" and "animals" sets, we varied the parameter δ. Since "covtype" has 44 binary dimensions, the corresponding values $\mathcal{M}_Q$ significantly underestimate the achieved data space reduction. Nevertheless, even they indicate very high degrees of the reduction.

Figure 8 shows the progress of the space reduction on "covtype" and "animals" at different levels of recursion for the density thresholds as in Table 1. For each level i, the plotted value $\mathcal{M}_i$ was computed counting all dense cells at levels $j < i$, if any, and all live regions at the level i. The reduction terminated or slowed down significantly after 5 or 6 levels.

The curves of Figure 8a suggest that deriving a γ partition using more extensive statistics about the data distribution in the given live region could be appropriate at the deeper levels of recursion. However, very high degrees of the reduction achieved at levels 4 and 5 suggest that, for the purposes of a stand-alone space reduction, the recursion can be terminated before it reaches deeper levels. Since the remaining false dense cells could still present problems for the effectiveness of data clustering, we did not restrict the depth of recursion in the experiments reported below.

6.2 Effectiveness and Efficiency of Clustering

As the measures of clustering effectiveness, we use the overall purity and entropy of clusters computed according to the formulas given in [23]. These are generic measures of clustering accuracy that can be applied to labeled data with

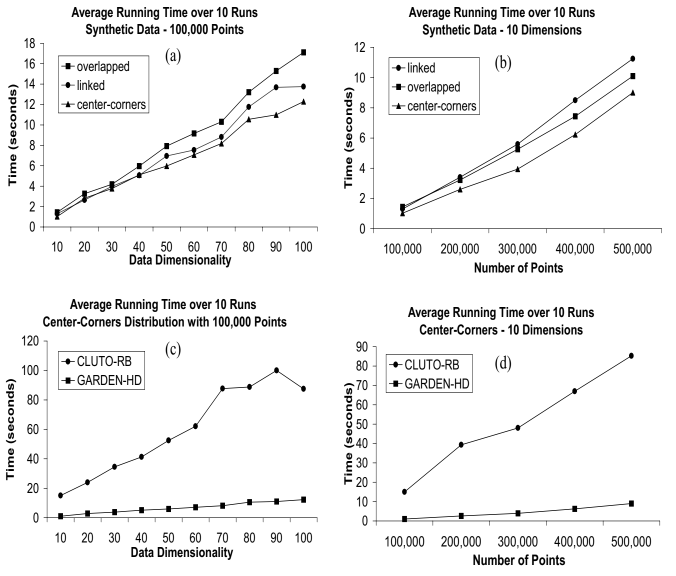

Figure 9: Average clustering times of GARDEN$_{\mathrm{HD}}$ (compared to CLUTO$_{\mathrm{RB}}$) on synthetic data.

class information. The purity of 1 and entropy of 0 indicate the best results. Depending on how the labels have been assigned, on many data sets, it may not possible to achieve the best purity and entropy. However, using these metrics, we can monitor how well GARDEN$_{HD}$ isolates pre-defined clusters in our synthetic data and evaluate the effectiveness of GARDEN$_{HD}$ as a general clustering method on both synthetic and real data. These metrics have also been used to evaluate our benchmark for comparison.

The results are compared to the default algorithm of CLUTO, a suite of clustering methods for high-dimensional data sets [12]. For a given L_C, this algorithm, referred to here as CLUTO$_{RB}$, derives L_C clusters via recursive bisections. Like GARDEN$_{HD}$, CLUTO$_{RB}$ is optimized for both efficiency and effectiveness, and it operates directly in the given multi-dimensional space. It is a general-purpose method known to be very efficient. These are some of the reasons that we adopted this method as benchmark for comparison. However, like other general-purpose algorithms, CLUTO$_{RB}$ still does not satisfy all requirements of our framework, in part because it takes the desired number of clusters as input.

One set of experiments involved synthetic high-dimensional data with the "center-corners" distribution as well as the "linked" and "overlapped" distributions shown earlier in Figure 7. In these sets, the data of every cluster was labeled with its own class, based on which we computed the overall purity and entropy of clusters. For GARDEN$_{HD}$, the selected density threshold δ was always set to the lowest estimated density of generated clusters.

Figures 9a and 9b display the average clustering times of GARDEN$_{HD}$ over 10 runs as data dimensionality or data

Table 2: Results of clustering the "covtype" set.

Method	ln(δ)	L_C	Purity	Entropy	Time (s)
GARDEN	80	84	0.651	0.389	33.173
GARDEN	100	112	0.652	0.389	33.514
CLUTOR	-	7	0.494	0.583	205.809
CLUTOR	-	84	0.518	0.548	568.425

Table 3: Results of clustering the "animals" set.

Method	ln(δ)	L_C	Purity	Entropy	Time (s)
GARDEN	160	1480	0.766	0.239	43.495
GARDEN	180	6208	0.999	0.001	44.095
CLUTOR	-	4	0.818	0.212	109.156
CLUTOR	-	10	0.908	0.144	223.511

volume grows.[2] The time was measured from the moment data was loaded in memory until the output began. This included the construction of the γ tree and the entire process of splitting and merging. In each scenario, GARDEN$_{HD}$ detected the exact number of clusters with the overall purity 1 and entropy 0. While Figure 9a reveals a close-to-linear performance degradation as dimensionality increases from 10 to 100, Figure 9b shows only a slight super-linear degradation as the data set grows from 100,000 to 500,000 points.

Figures 9c-d compare the clustering times of GARDEN$_{HD}$ and CLUTO$_{RB}$ on the "center-corners" sets as dimensionality and data volume increased. The input to CLUTO$_{RB}$ was the exact number of clusters (11). In the experiment of Figure 9c, GARDEN$_{HD}$ detected 11 clusters with the overall purity 1 and entropy 0, while CLUTO$_{RB}$ produced lower purities (between 0.839 and 0.53) and higher entropies (between 0.14 and 0.447) in a greater amount of time. Figure 9d gives the average clustering times of the two methods in a 10-dimensional space as the volume of data increased. While GARDEN$_{HD}$ detected all clusters correctly, CLUTO$_{RB}$ produced purities and entropies of about 0.84 and 0.14, respectively. Differences in the clustering times were about an order of magnitude.

Tables 2 and 3 give the results of GARDEN$_{HD}$ and CLUTO$_{RB}$ on "covtype" and "animals", respectively, for two different inputs. The column L_C gives the detected or selected number of clusters for GARDEN$_{HD}$ and CLUTO$_{RB}$, respectively. As in Figure 9, the recorded times represent the averages over 10 runs. For GARDEN$_{HD}$, the input density thresholds were the same as in Table 1. They were selected to achieve high purity and low entropy of clusters within the constraints for δ stated in Section 5.2. On "covtype", the selected number of clusters for CLUTO$_{RB}$ was 7 (the number of different classes in "covtype") and 84 (the number of clusters detected by GARDEN$_{HD}$ for $\delta = e^{80}$). On "animals", we set this number to 4 (the number of classes in the set) and 10 (the input that gives purity above 0.9 in the least amount of time).

Observe that, since the minimum distance μ is derived from δ, the number of clusters L_C detected by GARDEN$_{HD}$ may vary significantly with δ. On "covtype", GARDEN$_{HD}$ was not only faster but also more effective than CLUTO$_{RB}$. Despite very high density thresholds δ, which resulted in many clusters, GARDEN$_{HD}$ was faster than CLUTO$_{RB}$ on the "animals" data set as well.

[2]The times reported in the graphs and tables represent averages over 10 runs. We also computed the 95% confidence intervals of the timing results (not shown in the figure). Since the ratio of the confidence intervals to their respective results never exceed 4%, we consider our results statistically meaningful.

7. SUMMARY AND DISCUSSION

In this paper, we introduced GARDEN$_{HD}$, an effective and scalable algorithm for clustering data in multi-dimensional feature spaces. The technique is a hybrid of cell-based and density-based clustering. However, with its effective technique for data space reduction, it represents a different approach to data clustering that even in spaces with many dimensions can efficiently and accurately detect clusters of virtually any shape and spatial orientation.

Because of its efficiency, scalability, and the ability to effectively isolate areas with points, GARDEN$_{HD}$ is appropriate for any system that supports advanced content-based retrieval, including data-warehousing systems as well as multimedia and scientific databases. It can also be used to support new ways of storing and accessing massive data on hierarchical storage, like the organization with two levels of data clustering proposed in [16]. While GARDEN$_{HD}$ is designed to facilitate storage organizations that enable high-performance data access, experimental evidence indicates that it can be effective even as a general-purpose clustering algorithm.

The data space reduction technique of GARDEN$_{HD}$ relies on the properties of a new partitioning strategy, called γ. Using a density-based recursive application of γ partitioning, the technique can efficiently detect densely populated areas in the space. The properties of γ partitioning also enable efficient identification of live regions, which in huge high-dimensional spaces significantly reduce the amount of space that must be inspected.

The proposed data space reduction technique can be used not only to support effective reduction of the search space, but also as a data-reduction mechanism, alternative to (or in conjunction with) dimensionality reduction, compression, and sampling. It produces a concise representation that can provide data-mining applications useful insights into the data and its distribution. Like the agglomeration in GARDEN$_{HD}$, many data mining processes can be performed both efficiently and effectively on this representation alone.

The space-reduction technique can be adapted to operate like STING [20]. Like STING, the technique could be fully automatic and run in guaranteed O(N) time, requiring only a single pass over the data. However, unlike STING, the technique would be able to capture the essence of streaming data with not only low but also high dimensionality. Simple adaptations of the technique would also make it appealing for fast and scalable data classification and outlier detection.

Our future work includes the development of a new system for efficient and scalable retrieval of multi-dimensional data with explicit data clustering. When the system is complete, we will be able to evaluate the effectiveness of the proposed approach to clustering multi-dimensional data on storage in terms of its impact on retrieval performance.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. IIS-0312266.

8. REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan, “Automatic Subspace Clustering of High dimensional Data for Data Mining Applications,” *Proc. ACM SIGMOD Int. Conf. on Mngmt of Data*, 94–105, 1998.

[2] K. Chakrabarti and S. Mehrotra, “Local dimensionality Reduction: A New Approach to Indexing High dimensional Spaces,” *Proc. 26th Int. Conf. on Very Large Data Bases*, 89–100, 2000.

[3] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, 226–231, 1996.

[4] C. Faloutsos and K. Lin, “Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Databases,” *Proc. ACM SIGMOD Int. Conf. on Mngmt of Data*, 163–174, 1995.

[5] M. Halkidi, Y. Batistakis and M. Vazirgiannis, “Clustering Algorithms and Validity Measures,” *Proc. 13th Int. Conf. on Scientific and Stat. Database Management*, 3-22, 2001.

[6] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, California, 2001.

[7] A. Hinneburg and D. Keim, “An Efficient Approach to Clustering in Large Multimedia Databases with Noise,” *Proc. 4th Int. Conf. on Knowledge Discovery and Data Minging*, 58–65, 1998.

[8] A. Hinneburg and D.A. Keim, “Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-dimensional Clustering,” *Proc. 25th Int. Conf. on Very Large Data Bases*, 506–517, 1999.

[9] A.K. Jain, M.N. Murthy and P.J. Flynn, “Data Clustering: A Review,” *ACM Computing Surveys* 31(3):264–323, 1999.

[10] I.T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, New York, NY, 1986.

[11] K. Kailing, H.P. Kriegel and P. Kroger, “Density-Connected Subspace Clustering for High-dimensional Data,” *Proc. 4th SIAM Int. Conf. on Data Mining*, 246–256, 2004.

[12] G. Karypis, “CLUTO: A Clustering Toolkit,” Technical Report 02-017, Department of Computer Science, University of Minnesota, 2003.

[13] J.B. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” *Proc. 5th Symp. Math. Statist. and Probability*, Vol. 1, 281–297, 1967.

[14] H. Nagesh, S. Goil, and A.N. Choudhary, “A Scalable Parallel Subspace Clustering Algorithm for Massive Data Sets,” *Proc. Int. Conf. on Parallel Processing*, 477–486, 2000.

[15] R.T. Ng and J. Han, “Efficient and Effective Clustering Methods for Spatial Data Mining,” *Proc. 20th Int. Conf. on Very Large Data Bases*, 144–155, 1994.

[16] R. Orlandic, “Effective Management of Hierarchical Storage using Two Levels of Data Clustering,” *Proc. 20th IEEE/11th NASA Goddard Conf. on Mass Storage Systems and Techn.*, 270–279, 2003.

[17] R. Orlandic, J. Lukaszuk and C. Swietlik, “The Design of a Retrieval Technique for High-dimensional Data on Tertiary Storage,” *SIGMOD Record* 31(2):15–21, 2002.

[18] E.J. Otoo, A. Shoshani and S. Hwasng, “Clustering High dimensional Massive Scientific Datasets,” *Proc. 13th Int. Conf. on Scientific and Statistical Database Management*, 147–157, 2001.

[19] G. Sheikholeslami, S. Chatterjee and A. Zhang, “WaveCluster: A Multi-resolution Clustering Approach for Very Large Spatial Databases,” *Proc. 24th Int. Conf. on Very Large Data Bases*, 428–439, 1998.

[20] W. Wang, J. Yang and R.R. Muntz, “STING: A Statistical Information Grid Approach to Spatial Data Mining,” *Proc. 23rd Int. Conf. on Very Large Data Bases*, 186–195, 1997.

[21] S.B. Yao, “Approximating Block Accesses in Database Organizations,” *Communications of the ACM*, 20(4):260–261, 1977.

[22] T. Zhang, R. Ramakrishnan and M. Livny, “BIRCH: An Efficient Data Clustering Method for Very Large Databases,” *Proc. ACM SIGMOD Int. Conf. on Mngmt of Data*, 103–114, 1996.

[23] Y. Zhao and G. Karypis, “Criterion Functions for Document Clustering Experiments and Analysis,” Technical Report 01-40, Department of Computer Science, University of Minnesota, 2002.