Summarizing Spatial Data Streams Using ClusterHulls *

John Hershberger† Nisheeth Shrivastava‡ Subhash Suri‡

Abstract

We consider the following problem: given an on-line, possibly unbounded stream of two-dimensional points, how can we summarize its spatial distribution or *shape* using a small, bounded amount of memory? We propose a novel scheme, called *ClusterHull*, which represents the shape of the stream as a dynamic collection of convex hulls, with a total of at most m vertices, where m is the size of the memory. The algorithm dynamically adjusts both the number of hulls and the number of vertices in each hull to best represent the stream using its fixed memory budget. This algorithm addresses a problem whose importance is increasingly recognized, namely the problem of summarizing real-time data streams to enable on-line analytical processing. As a motivating example, consider habitat monitoring using wireless sensor networks. The sensors produce a steady stream of geographic data, namely, the locations of objects being tracked. In order to conserve their limited resources (power, bandwidth, storage), the sensors can compute, store, and exchange ClusterHull summaries of their data, without losing important geometric information. We are not aware of other schemes specifically designed for capturing shape information in geometric data streams, and so we compare ClusterHull with some of the best general-purpose clustering schemes such as CURE, k-median, and LSEARCH. We show through experiments that ClusterHull is able to represent the shape of two-dimensional data streams more faithfully and flexibly than the stream versions of these clustering algorithms.

1 Introduction

The extraction of meaning from data is perhaps the most important problem in all of science. Algorithms that can aid in this process by identifying useful structure are valuable in many areas of science, engineering, and information management. The problem takes many forms in different disciplines, but in many settings a *geometric abstraction* can be convenient: for instance, it helps formalize many informal but visually meaningful concepts such as similarity, groups, shape, etc. In many applications, geometric coordinates are a natural and integral part of data: e.g., locations of sensors in environmental monitoring, objects in location-aware computing, digital battlefield simulation, or meteorological data. Even when data have no intrinsic geometric association, many natural data analysis tasks such as clustering are best performed in an appropriate artificial coordinate space: e.g., data objects are mapped to points in some Euclidean space using certain attribute values, where similar objects (points) are grouped into spatial clusters for efficient indexing and retrieval. Thus we see that the problem of finding a simple characterization of a distribution known only through a collection of sample points is a fundamental one in many settings.

Recently there has been a growing interest in detecting patterns and analyzing trends in data that are generated continuously, often delivered in some fixed order and at a rapid rate. Some notable applications of such data processing include monitoring and surveillance using sensor networks, transactions in financial markets and stock exchanges, web logs and click streams, monitoring and traffic engineering of IP networks, telecommunication call records, retail and credit card transactions, and so on. Imagine, for instance, a surveillance application, where a remote environment instrumented by a wireless sensor network is being monitored through sensors that record the movement of objects (e.g., animals). The data gathered by each sensor can be thought of as a stream of two-dimensional points (geographic locations). Given the severe resource constraints of a wireless sensor network, it would be rather inefficient for each sensor to send its entire stream of raw data to a remote base sta-

*A partial summary of this work will be presented as a poster at ICDE '06, and represented in the proceedings by a three-page abstract.

†Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070, USA, and (by courtesy) Computer Science Department, University of California at Santa Barbara. john_hershberger@mentor.com.

‡Computer Science Department, University of California, Santa Barbara, CA 93106, USA. {nisheeth,suri}@cs.ucsb.edu. The research of Nisheeth Shrivastava and Subhash Suri was supported in part by National Science Foundation grants IIS-0121562 and CCF-0514738.

tion. Indeed, it would be far more efficient to compute and send a compact geometric summary of the trajectory. One can imagine many other remote monitoring applications like forest fire hazards, marine life, etc., where the shape of the observation point cloud is a natural and useful data summary. Thus, there are many sources of "transient" geometric data, where the key goal is to spot important trends and patterns, where only a small summary of the data can be stored, and where a "visual" summary such as shape or distribution of the data points is quite valuable to an analyst.

A common theme underlying these data processing applications is the continuous, real-time, large-volume, transient, single-pass nature of data. As a result, *data streams* have emerged as an important paradigm for designing algorithms and answering database queries for these applications. In the data stream model, one assumes that data arrive as a continuous stream, in some arbitrary order possibly determined by an adversary; the total size of the data stream is quite large; the algorithm may have memory to store only a tiny fraction of the stream; and any data not explicitly stored are essentially lost. Thus, data stream processing necessarily entails *data reduction*, where most of the data elements are discarded and only a small representative sample is kept. At the same time, the patterns or queries that the applications seek may require knowledge of the entire history of the stream, or a large portion of it, not just the most recent fraction of the data. The lack of access to full data significantly complicates the task of data analysis, because patterns are often hidden, and easily lost unless care is taken during the data reduction process. For simple database aggregates, sub-sampling can be appropriate, but for many advanced queries or patterns, sophisticated synopses or summaries must be constructed. Many such schemes have recently been developed for computing quantile summaries [21], most frequent or top-k items [23], distinct item counts [3, 24], etc.

When dealing with geometric data, an analyst's goal is often not as precisely stated as many of these *numerically-oriented* database queries. The analyst may wish to understand the general structure of the data stream, look for unusual patterns, or search for certain "qualitative" anomalies before diving into a more precisely focused and quantitative analysis. The "shape" of a point cloud, for instance, can convey important qualitative aspects of a data set more effectively than many numerical statistics. In a stream setting, where the data must be constantly discarded and compressed, special care must be taken to ensure that the sampling faithfully captures the overall shape of the point distribution.

Shape is an elusive concept, which is quite challenging even to define precisely. Many areas of computer science, including computer vision, computer graphics, and computational geometry deal with representation, matching and extraction of shape. However, techniques in those areas tend to be computationally expensive and unsuited for data streams. One of the more successful techniques in processing of data streams is clustering. The clustering algorithms are mainly concerned with identifying dense groups of points, and are not specifically designed to extract the boundary features of the cluster groups. Nevertheless, by maintaining some sample points in each cluster, one can extract some information about the geometric shape of the clusters. We will show, perhaps unsurprisingly, that ClusterHull, which explicitly aims to summarize the geometric shape of the input point stream using a limited memory budget, is more effective than general-purpose stream clustering schemes, such as CURE, k-median and LSEARCH.

1.1 ClusterHull

Given an on-line, possibly unbounded stream of two-dimensional points, we propose a scheme for summarizing its spatial distribution or *shape* using a small, bounded amount of memory m. Our scheme, called *ClusterHull*, represents the shape of the stream as a dynamic collection of convex hulls, with a total of at most m vertices. The algorithm dynamically adjusts both the number of hulls and the number of vertices in each hull to represent the stream using its fixed memory budget. Thus, the algorithm attempts to capture the shape by decomposing the stream of points into groups or clusters and maintaining an approximate convex hull of each group. Depending on the input, the algorithm adaptively spends more points on clusters with complex (potentially more interesting) boundaries and fewer on simple clusters. Because each cluster is represented by its convex hull, the ClusterHull summary is particularly useful for preserving such geometric characteristics of each cluster as its boundary shape, orientation, and volume. Because hulls are objects with spatial extent, we can also maintain additional information such as the number of input points contained within each hull, or their approximate *data density* (e.g., population divided by the hull volume). By shading the hulls in proportion to their density, we can then compactly convey a simple visual representation of the data distribution. By contrast, such information seems difficult to maintain in stream clustering schemes, because the cluster centers in those schemes

constantly move during the algorithm.

For illustration, in Figure 1 we compare the output of our ClusterHull algorithm with those produced by two popular stream-clustering schemes, k-median [19] and CURE [20]. The top row shows the input data (left), and output of ClusterHull (right) with memory budget set to $m = 45$ points. The middle row shows outputs of k-median, while the bottom row shows the outputs of CURE. One can see that both the boundary shapes and the densities of the point clusters are quite accurately summarized by the cluster hulls.

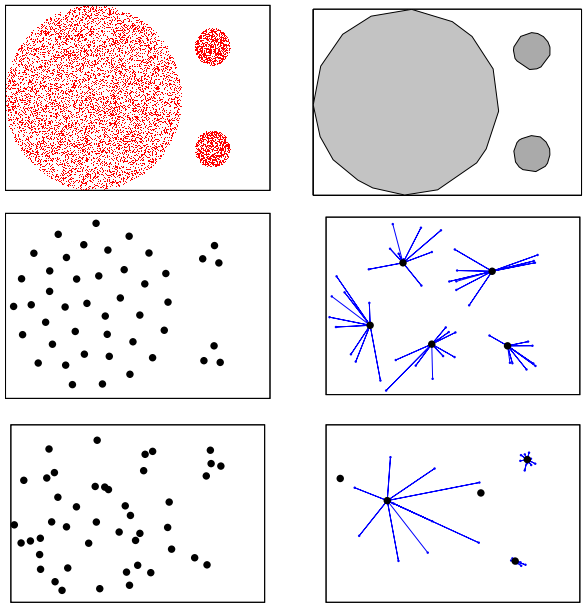


Figure 1: The top row shows the input data (left) and the output of ClusterHull (right) with memory budget of $m = 45$. The hulls are shaded in proportion to their estimated point density. The middle row shows two different outputs of the stream k**-medians** algorithm, with $m = 45$: in one case (left), the algorithm simply computes $k = 45$ cluster centers; in the other (right), the algorithm computes $k = 5$ centers, but maintains 9 (random) sample points from the cluster to get a rough approximation of the cluster geometry. (This is a simple enhancement implemented by us to give more expressive power to the k-median algorithm.) Finally, the bottom row shows the outputs of **CURE**: in the left figure, the algorithm computes $k = 45$ cluster centers; in the right figure, the algorithm computes $k = 5$ clusters, with $c = 9$ samples per cluster. CURE has a tunable shrinkage parameter, α, which we set to 0.4, in the middle of the range suggested by its authors [20].

We implemented ClusterHull and experimented with both synthetic and real data to evaluate its performance. In all cases, the representation by ClusterHull appears to be more information-rich than those by clustering schemes such as CURE, k-medians, or LSEARCH, even when the latter are enhanced with some simple mechanisms to capture cluster shape. Thus, our general conclusion is that ClusterHull can be a useful tool for summarizing geometric data streams.

ClusterHull is computationally efficient, and thus well-suited for streaming data. At the arrival of each new point, the algorithm must decide whether the point lies in one of the existing hulls (actually, within a certain ring around each hull), and possibly merge two existing hulls. With appropriate data structures, this processing can be done in amortized time $O(\log m)$ per point.

ClusterHull is a general paradigm, which can be extended in several orthogonal directions and adapted to different applications. For instance, if the input data are *noisy*, then covering all points by cluster hulls can lead to poor shape results. We propose an *incremental cleanup* mechanism, in which we periodically discard light-weight hulls, that deals with noise in the data very effectively. Similarly, the performance of a shape summary scheme can depend on the order in which input is presented. If points are presented in a bad order, the ClusterHull algorithm may create long, skinny, inter-penetrating hulls early in the stream processing. We show that a *period-doubling cleanup* is effective in correcting the effects of these early mistakes. When there is spatial coherence within the data stream, our scheme is able to exploit that coherence. For instance, imagine a point stream generated by a sensor field monitoring the movement of an *unknown* number of vehicles in a two-dimensional plane. The data naturally cluster into a set of spatially coherent trajectories, which our algorithm is able to isolate and represent more effectively than general-purpose clustering algorithms.

1.2 Related Work

Inferring shape from an unordered point cloud is a well-studied problem that has been considered in many fields, including computer vision, machine learning, pattern analysis, and computational geometry [4, 10, 11, 26]. However, the classical algorithms from these areas tend to be computationally expensive and require full access to data, making them unsuited for use in a data stream setting.

An area where significant progress has occurred on stream algorithms is *clustering*. Our focus is some-

what different from classical clustering—we are mainly interested in low-dimensional data and capturing the "surface" or boundary of the point cloud, while clustering tends to focus on the "volume" or density and moderate and large dimensions. While classical clustering schemes of the past have focused on cluster centers, which work well for spherical clusters, some recent work has addressed the problem of non-spherical clusters, and tried to pay more attention to the geometry of the clusters. Still this attention to geometry does not extend to the *shape* of the boundary.

Our aim is not to exhaustively survey the clustering literature, which is immense and growing, but only to comment briefly on those clustering schemes that could potentially be relevant to the problem of summarizing shape of two- or three-dimensional point streams. Many well-known clustering schemes (e.g., [5, 7, 16, 25]) require excessive computation and require multiple passes over the data, making them unsuited for our problem setting. There are machine-learning based clustering schemes [12, 13, 27], that use classification to group items into clusters. These methods are based on statistical functions, and not geared towards shape representation. Clustering algorithms based on spectral methods [8, 14, 18, 28] use the singular value decomposition on the similarity graph of the data, and are good at clustering statistical data, especially in high dimensions. We are unaware of any results showing that these methods are particularly effective at capturing boundary shapes, and, more importantly, streaming versions of these algorithms are not available. So, we now focus on clustering schemes that work on streams and are designed to capture some of the geometric information about clusters.

One of the popular clustering schemes for large data sets is BIRCH [30], which also works on data streams. An extension of BIRCH by Aggarwal et al. [2] also computes multi-resolution clusters in evolving streams. While BIRCH appears to work well for spherical-shaped clusters of uniform size, Guha et al. [20] experimentally show that it performs poorly when the data are clustered into groups of unequal sizes and different shapes. The CURE clustering scheme proposed by Guha et al. [20] addresses this problem, and is better at identifying non-spherical clusters. CURE also maintains a number of sample points for each cluster, which can be used to deduce the geometry of the cluster. It can also be extended easily for streaming data (as noted in[19]). Thus, CURE is one of the clustering schemes we compare against ClusterHull.

In [19], Guha et al. propose two stream variants of k-center clustering, with provable theoretical guarantees as well as experimental support for their performance. The stream k-median algorithm attempts to minimize the sum of the distances between the input points and their cluster centers. Guha et al. [19] also propose a variant where the number of clusters k can be relaxed during the intermediate steps of the algorithm. They call this algorithm LSEARCH (local search). Through experimentation, they argue that the stream versions of their k-median and LSEARCH algorithms produce better quality clusters than BIRCH, although the latter is computationally more efficient. Since we are chiefly concerned with the quality of the shape, we compare the output of ClusterHull against the results of k-median and LSEARCH (but not BIRCH).

1.3 Organization

The paper is organized in seven sections. Section 2 describes the basic algorithm for computing cluster hulls. In Section 3 we discuss the cost function used in refining and unrefining our cluster hulls. Section 4 provides extensions to the basic ClusterHull algorithm. In Sections 5 and 6 we present some experimental results. We conclude in Section 7.

2 Representing Shape as a Cluster of Hulls

We are interested in simple, highly efficient algorithms that can identify and maintain *bounded-memory approximations* of a stream of points. Some techniques from computational geometry appear especially well-suited for this. For instance, the *convex hull* is a useful shape representation of the *outer boundary* of the whole data stream. Although the convex hull accurately represents a convex shape with an arbitrary aspect ratio and orientation, it loses all the internal details. Therefore, when the points are distributed non-uniformly within the convex hull, the outer hull is a poor representation of the data.

Clustering schemes, such as k-medians, partition the points into groups that may represent the distribution better. However, because the goal of many clustering schemes is typically to minimize the maximum or the sum of distance functions, there is no explicit attention given to the shape of clusters—each cluster is conceptually treated as a ball, centered at the cluster center. Our goal is to mediate between the two extremes offered by the convex hull and k-medians. We would like to combine the best features of the convex hull—its ability to represent convex shapes with any

aspect ratio accurately—with those of ball-covering approximations such as k-medians—their ability to represent nonconvex and disconnected point sets. With this motivation, we propose the following measure for representing the shape of a point set under the bounded memory constraint.

> *Given a two-dimensional set of N points, and a memory budget of m, where $m \ll N$, compute a set of convex hulls such that (1) the collection of hulls uses at most m vertices, (2) the hulls together cover all the points of S, and (3) the total area covered by the hulls is minimized.*

Intuitively, this definition interpolates between a single convex hull, which potentially covers a large area, and k-medians clustering, which fails to represent the shape of individual clusters accurately. Later we will relax the condition of "covering all the points" to deal with *noisy data*—in the relaxed problem, *a constant fraction* of the points may be dropped from consideration. But the general goal will remain the same: to compute a set of convex hulls that attempts to cover the important geometric features of the data stream using least possible area, under the constraint that the algorithm is allowed to use at most m vertices.

2.1 Geometric approximation in data streams

Even the classical convex hull (outer boundary) computation involves some subtle and nontrivial issues in the data stream setting. What should one do when the number of extreme vertices in the convex hull exceeds the memory available? Clearly, some of the extreme vertices must be dropped. But which ones, and how shall we measure the *error* introduced in this approximation? This problem of summarizing the convex hull of a point stream using a fixed memory m has been studied recently in computational geometry and data streams [1, 6, 9, 17, 22]. An adaptive sampling scheme proposed in [22] achieves an optimal memory-error tradeoff in the following sense: given memory m, the algorithm maintains a hull that (1) lies within the true convex hull, (2) uses at most m vertices, and (3) approximates the true hull well—any input point not in the computed hull lies within distance $O(D/m^2)$ of the hull, where D is the diameter of the point stream. Moreover, the error bound of $O(D/m^2)$ is the best possible in the worst case.

In our problem setting, we will maintain not one but many convex hulls, depending on the geometry of the stream, with each hull roughly corresponding to a cluster. Moreover, the locations of these hulls are not determined *a priori*—rather, as in k-medians, they are dynamically determined by the algorithm. Unlike k-medians clusters, however, each hull can use a different fraction of the available memory to represent its cluster boundary. One of the key challenges in designing the ClusterHull algorithm is to formulate a good policy for this memory allocation. For this we will introduce a cost function that the various hulls use to decide how many hull vertices each gets. Let us first begin with an outline of our scheme.

2.2 The basic algorithm

The available memory m is divided into two pools: a fixed pool of k groups, each with a constant number of vertices; and a shared pool of $O(k)$ points, from which different cluster hulls draw additional vertices. The number k has the same rôle as the parameter fed to k-medians clustering—it is set to some number at least as large as the number of native clusters expected in the input. (Thus our representation will maintain a more refined view of the cluster structure than necessary, but simple post-processing can clean up the unnecessary sub-clustering.) The exact constants in this division are tunable, and we show their effect on the performance of the algorithm through experimentation. For the sake of concreteness, we can assume that each of the k groups is initially allocated 8 vertices, and the common pool has a total of $8k$ vertices. Thus, if the available memory is m, then we must have $m \geq 16k$.

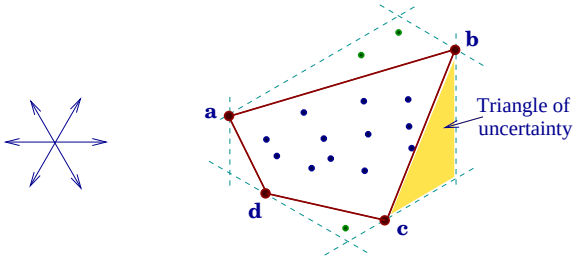


Figure 2: An approximate hull, with 6 sampling directions. The sample hull's vertices are a, b, c, d.

Our algorithm approximates the convex hull of each group by its extreme vertices in selected (sample) directions: among all the points assigned to this cluster group, for each sample direction, the algorithm retains the extreme vertex in that direction. See Figure 2 for an example. Each edge of this sampled hull supports what we call an *uncertainty triangle*—the triangle formed by the edge and the tangents at the two

endpoints of the edge in the sample directions for which those endpoints are extreme. A simple but important property of the construction is that the boundary of the true convex hull is sandwiched in the ring of uncertainty triangles defined by the edges of the computed hull. See Figure 3 for an illustration. The extremal directions are divided into two sets, one containing uniformly-spaced fixed directions, corresponding to the initial endowment of memory, and another containing adaptively chosen directions, corresponding to additional memory drawn from the common pool. The adaptive directions are added incrementally, bisecting previously chosen directional intervals, to minimize the error of the approximation.

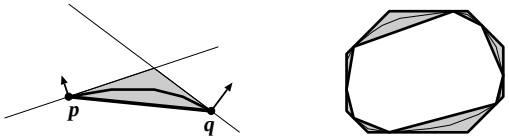


Figure 3: The true hull is sandwiched in a ring of uncertainty triangles.

Each hull has an individual cost associated with it, and the whole collection of k hulls has a total cost that is the sum of the individual costs. Our goal is to choose the cost function such that minimizing the total cost leads to a set of approximate convex hulls that represent the shape of the point set well. Furthermore, because our minimization is performed on-line, assigning each new point in the stream to a convex hull when the point arrives, we want our cost function to be *robust*: as much as possible, we want it to reduce the chance of assigning early-arriving points to hulls in a way that forces late-arriving points to incur high cost. We leave the technical details of our choice of the cost function to the following section.

Let us now describe the high-level organization of our algorithm. Suppose that the current point set S is partitioned among k convex hulls $H_1, \ldots, H_k$. The cost of hull H_i is $w(H_i)$, and the total cost of the partition $\mathcal{H} = \{H_1, \ldots, H_k\}$ is $w(\mathcal{H}) = \sum_{H \in \mathcal{H}} w(H)$. We process each incoming point p with the following algorithm:

Algorithm ClusterHull

if p is contained in any $H \in \mathcal{H}$, or in the ring of uncertainty triangles for any such H, **then**
  Assign p to H without modifying H.
else
  Create a new hull containing only p and add it to $\mathcal{H}$.
  if $|\mathcal{H}| > k$ **then**
    Choose two hulls $H, H' \in \mathcal{H}$ such that merging H and H' into a single convex hull will result in the minimum increase to $w(\mathcal{H})$.
    Remove H and H' from $\mathcal{H}$, merge them to form a new hull H^*, and put that into $\mathcal{H}$.
    If H^* has an uncertainty triangle over either edge joining points of the former H and H' whose height exceeds the previous maximum uncertainty triangle height, refine (repeatedly bisect) the angular interval associated with that uncertainty triangle by choosing new adaptive directions until the triangle height is less than the previous maximum.
    while the total number of adaptive directions in use exceeds ck
      Unrefine (discard one of the adaptive directions for some $H \in \mathcal{H}$) so that the uncertainty triangle created by unrefinement has minimum height.

The last two steps (refinement and unrefinement) are technical steps for preserving the approximation quality of the convex hulls that were introduced in [22]. The key observation is that an uncertainty triangle with "large height" leads to a poor approximation of a convex hull. Ideally, we would like uncertainty triangles to be flat. The height of an uncertainty triangle is determined by two key variables: the length of the convex hull edge, and the angle-difference between the two sampling directions that form that triangle. More precisely, consider an edge $\overline{pq}$. We can assume that the extreme directions for p and q, namely, θ_p and θ_q, point toward the same side of $\overline{pq}$, and hence the intersection of the supporting lines projects perpendicularly onto $\overline{pq}$. Therefore the height of the uncertainty triangle is at most the edge length $\ell(\overline{pq})$ times the tangent of the smaller of the angles between $\overline{pq}$ and the supporting lines. Observe that the sum of these two angles equals the angle between the directions θ_p and θ_q. If we define $\theta(\overline{pq})$ to be $|\theta_p - \theta_q|$, then the height of the uncertainty triangle at $\overline{pq}$ is at most $\ell(\overline{pq}) \cdot \tan(\theta(\overline{pq})/2)$, which is closely approximated by

$$\frac{\ell(\overline{pq}) \cdot \theta(\overline{pq})}{2}. \tag{2.1}$$

This formula forms the basis for adaptively choosing new sampling directions: we devote more sampling directions to cluster hull edges whose uncertainty triangles have large height. Refinement is the process of introducing a new sampling direction that bisects two consecutive sampling directions; unrefinement is the converse of this process. The analysis in [22] showed

that if a single convex hull is maintained using $m/2$ uniformly spaced sampling directions, and $m/2$ adaptively chosen directions (using the policy of minimizing the maximum height of an uncertainty triangle), then the maximum distance error between true and approximate hulls is $O(D/m^2)$. Because in ClusterHull we share the refinement directions among k different hulls, we choose them to minimize the global maximum uncertainty triangle height explicitly. We point out that the allocation of adaptive directions is independent of the cost function $w(\mathcal{H})$. The cost function guides the partition into convex hulls; once that choice is made, we allocate adaptive directions to minimize the error for that partition. One could imagine making the assignment of adaptive directions dependent on the cost function, but for simplicity we have chosen not to do so.

3 Choosing a Cost Function

In this section we describe the cost function we apply to the convex hulls that ClusterHull maintains. We discuss the intuition behind the cost function, experimental support for that intuition, and variants on the cost function that we considered.

The *α-hull* is a well-known structure for representing the shape of a set of points [15]. It can be viewed as an extension of the convex hull in which half-planes are replaced by the complements of fixed-radius disks (i.e., the regions outside the disks). In particular, the convex hull is the intersection of all half-planes containing the point set, and the α-hull is the intersection of all disk-complements with radius ρ that contain the point set.[1] See Figure 4 for examples of the convex hull and α-hull on an L-shaped point set. The α-hull minimizes the area of the shape that covers the points, subject to the radius constraint on the disks.

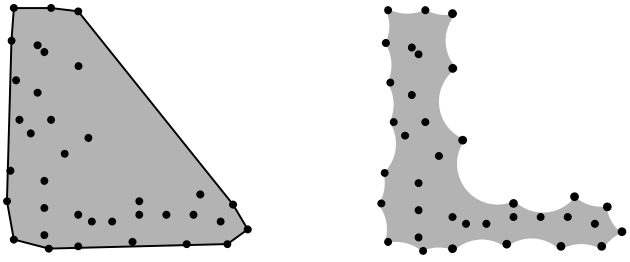


Figure 4: Shape representations for a set of points: (left) convex hull, (right) α-hull.

[1] In the definition of α-hulls, the disk radius $\rho = 1/|\alpha|$, and $\alpha \leq 0$, but we are not concerned with these technical details.

The α-hull is not well suited to represent the shape of a stream of points, because an unbounded number of input points may appear on the boundary of the shape. Our goal of covering the input points with bounded-complexity convex hulls of minimum total area is an attempt to mimic the modeling power of the α-hull in a data stream setting.

Although our goal is to minimize the total area of our convex hull representation, we use a slightly more complex function as the cost of a convex hull H:

$$w(H) = \texttt{area}(H) + \mu \cdot (\texttt{perimeter}(H))^2. \tag{3.2}$$

Here μ is a constant, chosen empirically as described below. Note that the perimeter is squared in this expression to match units: if the perimeter term entered linearly, then simply changing the units of measurement would change the relative importance of the area and perimeter terms, which would be undesirable.

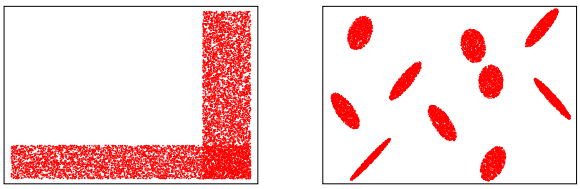


Figure 5: Input distributions: L-shaped and *ellipses*.

We want to minimize total area, and so defining $w(H) = \texttt{area}(H)$ seems natural; however, this proves to be infeasible in a stream setting. If a point set has only two points, the area of its convex hull is zero; thus all such hulls have the same cost. The first $2k$ points that arrive in a data stream are paired up into k two-point convex hulls, each with cost zero, and the pairing will be arbitrary. In particular, some convex hulls are likely to cross natural cluster boundaries. When these clusters grow as more points arrive, they will have higher cost than the optimal hulls that would have been chosen by an off-line algorithm. This effect is clearly visible in the clusters produced by our algorithm in Figure 6 (right) for the *ellipses* data set of Figure 5 (right). By contrast, the L-shaped distribution of Figure 5 (left) is recovered well using the area cost function, as shown in Figure 6 (left).

We can avoid the tendency of the area cost to create long thin needles in the early stages of the stream by minimizing the perimeter. If we choose $w(H) = \texttt{perimeter}(H)$, then the well-separated clusters of the *ellipses* data set are recovered perfectly, even when the points arrive on-line—see Figure 7 (right).

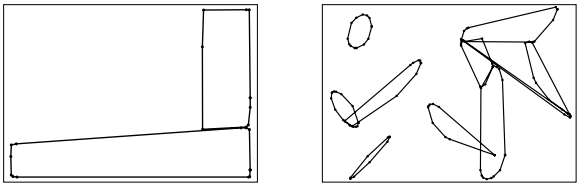


Figure 6: With the area cost function, ClusterHull faithfully recovers the L-shaped distribution of points. But it performs poorly on a set of $n = 10,000$ points distributed among ten elliptical clusters; it merges pairs of points from different groups and creates intersecting hulls.

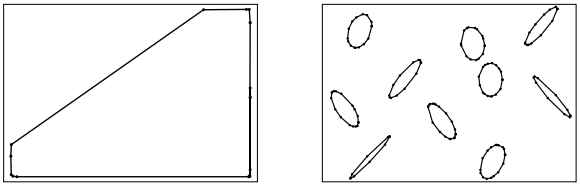


Figure 7: With the perimeter cost function, ClusterHull faithfully recovers the disjoint elliptical clusters, but performs poorly on the L-shaped distribution.

However, as the poor recovery of the L distribution shows (Figure 7 (left)), the perimeter cost has its own liabilities. The total perimeter of two hulls that are relatively near each other can often be reduced by merging the two into one. Furthermore, merging two large hulls reduces the perimeter more than merging two similar small ones, and so the perimeter cost applied to a stream often results in many small hulls and a few large ones that contain multiple "natural" clusters.

We need to incorporate both area and perimeter into our cost function to avoid the problems shown in Figures 6 and 7. Because our overall goal is to minimize area, we choose to keep the area term primary in our cost function (Equation 3.2). In that function $(\texttt{perimeter}(H))^2$ is multiplied by a constant μ, which is chosen to adjust the relative importance of area and perimeter in the cost. Experimentation shows that choosing $\mu = 0.05$ gives good shape reconstruction on a variety of inputs. With μ substantially smaller than 0.05, the perimeter effect is not strong enough, and with μ greater than 0.1, it is too strong. (Intuitively, we want to add just enough perimeter dependence to avoid creating needle convex hulls in the early stages of the stream.)

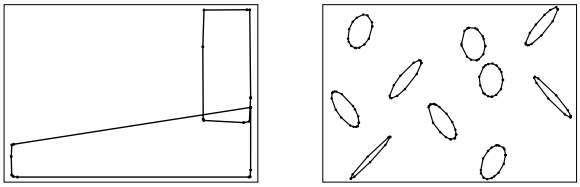


Figure 8: With the combined area and perimeter cost function, the algorithm ClusterHull recovers both the ellipse and L distributions. The choice of $\mu = 0.05$ gives good shape reconstruction.

We can understand the combined area-perimeter cost by modeling it as the area of a fattened convex hull. If we let $\rho = \mu \cdot \texttt{perimeter}(H)$, we see that the area-perimeter cost (3.2) is very close to the area obtained by fattening H by ρ. The true area is $\texttt{area}(H)+\rho\cdot\texttt{perimeter}(H)+\pi\rho^2 = \texttt{area}(H)+\rho^2(\frac{1}{\mu}+\pi)$; if μ is small, then $1/\mu$ is relatively large compared to π, and the extra $\pi\rho^2$ term is not very significant.

Because the cost (3.2) may fatten long thin clusters more than is desirable, we also experimented with replacing the constant μ in (3.2) by a value inversely related to the aspect ratio of H. The aspect ratio of H is $\texttt{diam}(H)/\texttt{width}(H) = \Theta((\texttt{perimeter}(H))^2/\texttt{area}(H))$. Thus if we simply replaced μ by $1/\texttt{aspectRatio}(H)$ in (3.2), we would essentially obtain the area cost. We compromised by using the cost

$$w(H) = \texttt{area}(H) + \mu \cdot (\texttt{perimeter}(H))^2/(\texttt{aspectRatio}(H))^x$$

for various values of x ($x = 0.5$, $x = 0.1$). The aspect ratio is conveniently approximated as $(\texttt{perimeter}(H))^2/\texttt{area}(H)$, since the quantities in that expression are already maintained by our convex hull approximation. Except in extreme cases, the results with this cost function were not enough different from the basic area-perimeter cost to merit a separate figure.

The cost (3.2) fattens each hull by a radius proportional to its own perimeter. This is appropriate if the clusters have different natural scales and we want to fatten each according to its own dimensions. However, in our motivating structure the α-hull, a uniform radius is used to define all the clusters. To fatten hulls uniformly, we could use the weight function

$$w(H) = \texttt{area}(H) + \rho \cdot \texttt{perimeter}(H) + \pi\rho^2.$$

However, the choice of the fattening radius ρ is problematic. We might like to choose ρ such that α-hulls

defined using radius-ρ disks form exactly k clusters, but then the optimum value of ρ would decrease and increase as the stream points arrived. We can avoid these difficulties by sticking to the simpler cost of definition (3.2).

4 Extensions and Enhancements

In this section we discuss how to enhance the basic ClusterHull algorithm to improve the quality of shape representation.

4.1 Spatial incoherence and period-doubling cleanup

In many data streams the arriving points are ordered arbitrarily, possibly even adversarily. The ClusterHull scheme (and indeed any on-line clustering algorithm) is vulnerable to early errors, in which an early-arriving point is assigned to a hull that later proves to be the wrong one.

Figure 9 (left) shows a particularly bad input consisting of five thin parallel stripes. We used ClusterHull with $\mu = 0.05$ to maintain five hulls, with the input points ordered randomly. A low density sample from the stripe distribution (such as a prefix of the stream) looks to the algorithm very much like uniformly distributed points. Early hull merges combine hulls from different stripes, and the ClusterHull algorithm cannot recover from this mistake. See Figure 9 (right).

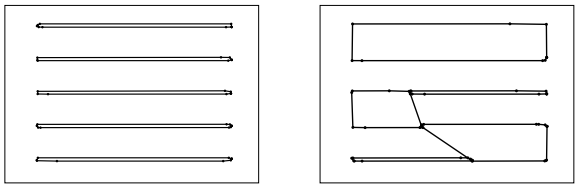


Figure 9: Processing the *stripes* input (left) in random order leads to errors for our algorithm (right).

If the input data arrive in random order, the idea of *period-doubling cleanup* may help identify and amplify the true clusters. The idea is to process the input stream in *rounds* in which the number of points processed doubles in each round. At the end of each round we identify low density hulls and discard them—these likely group points from several true clusters. The dense hulls are retained from round to round, and are allowed to grow.

Formally, the period-doubling cleanup operates as follows: For each $H \in \mathcal{H}$ we maintain the number of points it represents, denoted by $\texttt{count}(H)$. The density of any hull H is $\texttt{density}(H) = \texttt{count}(H)/\texttt{area}(H)$. The algorithm also maintains an approximate convex hull G of all the input points. After each round, it discards from $\mathcal{H}$ every hull H for which any of the following holds:

- $\texttt{count}(H) < \delta \cdot N/k$
- $\texttt{density}(H) < \texttt{density}(G)$
- $\texttt{density}(H) < \frac{1}{2} \cdot \texttt{median}\{\texttt{density}(\texttt{A}) : \texttt{A} \in \mathcal{H}\}$

Here N is the number of points seen so far. In our experiments, we set the tunable parameter δ to 0.1.

The first test takes care of hulls with a very small number of tightly clustered points (these may have high densities because of their smaller area, and will not be caught by density pruning). The second test discards hulls that have less than average density. The intuition is that each cluster should be at least as dense as the entire input space (otherwise it is not an interesting cluster). In case the points are distributed over a very large area, but the individual clusters are very compact, the average density may not be very helpful for discarding hulls. Instead, we should discard hulls that have low densities relative to other hulls in the data structure; the third test takes care of this case—it discards any hull with density significantly less than the median density.

Figure 10 (left) shows the result of period-doubling cleanup on the *stripes* distribution; the sparse hulls that were initially found have been discarded and five dense hulls have been correctly computed. We note that, with the same amount of memory, neither CURE nor the k-median clustering is able to represent the *stripes* distribution well (cf. Figure 10). Our experiments show that applying period-doubling cleanup helps improve clustering on almost all data sets.

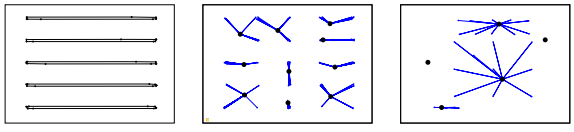


Figure 10: Period-doubling cleanup (left) on ClusterHull corrects the errors in the *stripes* distribution; the middle figure shows the output of k-medians, and the right figure shows the output of CURE.

4.2 Noisy data and incremental cleanup

Sampling error and outliers cause difficulty for nearly all clustering algorithms. Likewise, a few outliers can adversely affect the ClusterHull shape summary. An algorithm needs some way to distinguish between dense regions of the input distribution (the true clusters) and sparse ones (noise). In this section, we propose an *incremental cleanup* mechanism that can improve the performance of our algorithm in the presence of noise. Both the period-doubling and the incremental cleanup are inspired by sampling techniques used in frequency estimation in streams. In particular, period-doubling is inspired by sticky sampling and incremental cleanup is inspired by lossy counting [23]. The incremental cleanup also processes the input in rounds, but the rounds do not increase in length. This is because noise is not limited to the beginning of the input; if we increased the round length, all the hulls would be corrupted by noisy points. Instead, we fix the size of each round depending on the (estimated) noise in the input.

Specifically, the incremental cleanup assumes that the input stream consists of points drawn randomly from a fixed distribution, with roughly $(1-\epsilon)N$ of them belonging to high density clusters and ϵN of them being low density noise. The expected noise frequency ϵ affects the quality of the output. We can estimate it conservatively if it is unknown. The idea is to set the value of δ to be roughly equal to ϵ, and process the input in rounds of $k/(2\epsilon)$ points. The logic is that in every round, only about $k/2$ hulls will be corrupted by noisy points, still leaving half of the hulls untouched and free to track the true distribution of the input. If we set k to be more than twice the expected number of natural clusters in the input, we obtain a good representation of the clusters.

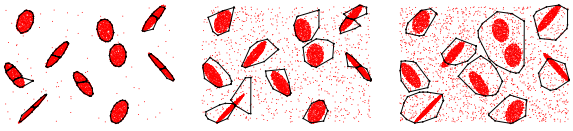
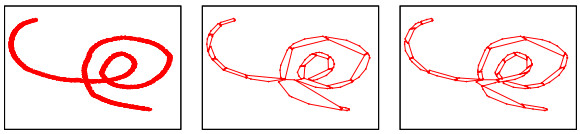


Figure 11: Incremental cleanup, with estimated noise frequency $\epsilon = 0.1$, applied to distributions with 1%, 10%, and 20% actual background noise.

This scheme propagates the good hulls (those with high density and high cardinality) from one round of the algorithm to the next, while discarding hulls that are sparse or belong to outliers. See Figure 11 for an example of how this scheme identifies true clusters and discards noisy regions. Of course, if noise is underestimated significantly (Figure 11 (right)), the quality of the cluster hulls suffers.

4.3 Spatial coherence and trajectory tracking

Section 4.1 considered spatially incoherent input streams. If the input *is* spatially coherent, as occurs in some applications, ClusterHull performs particularly well. If the input stream consists of locations reported by sensors detecting some moving entity (a light pen on a tablet, a tank in a battlefield, animals in a remote habitat), our algorithm effectively finds a covering of the trajectory by convex "lozenges." The algorithm also works well when there are multiple simultaneous trajectories to represent, as might occur when sensors track multiple independent entities. If the stripes of Figure 9 are fed to the algorithm in left-to-right order they are recovered perfectly; likewise in Figure 12 a synthetic trajectory is represented accurately.



(a) (b) (c)

Figure 12: Input along a trajectory in (a); the shape is recovered well using $m = 100$ in (b), and using $m = 150$ in (c).

4.4 Density estimation and display

Stream versions of traditional clustering schemes (including k-median and CURE) do not include an estimate of the density of points associated with each cluster center, whereas each cluster hull H can easily maintain $\texttt{count}(H)$. As in Section 4.1, this gives an estimate of the density of the points in each hull. If this information is displayed graphically (cf. Figure 13) it conveys more insight about the distribution of the input data than does the simple cluster-center output or even cluster-sample output.

5 Implementation and Experiments

We implemented the convex hull algorithm of [22] and the ClusterHull algorithm on top of it. The

convex hull algorithm takes logarithmic time per inserted point, on the average, but our ClusterHull implementation is more simple-minded, optimized more for ease of implementation than for runtime performance. The bottleneck in our implementation is neighborhood queries/point location, taking time proportional to the number of hulls. By storing the hull edges in a quad-tree, we could speed up these operations to $O(\log m)$ time.

When a new point arrives, we must check which hull it belongs to, if any. Using a quad-tree, this reduces to a logarithmic-time search, followed by logarithmic-time point-in-polygon tests with an expected constant number of hulls. Each new hull H must compute its optimum merge cost—the minimum increment to $w(\mathcal{H})$ caused by merging H with another hull. On average, this increment is greater for more distant hulls. Using the quad-tree we can compute the increment for $O(1)$ nearby hulls first. Simple lower bounds on the incremental cost of distant merges then let us avoid computing the costs for distant hulls. Computing the incremental cost for a single pair of hulls reduces to computing tangents between the hulls, which takes logarithmic time [22].

Merging hulls eliminates some vertices forever, and so we can charge the time spent performing the merge to the deleted vertices. Thus a careful implementation of the ClusterHull algorithm would process stream points in $O(\log m)$ amortized time per point, where m is the total number of hull vertices.

In the remainder of this section we evaluate the performance of our algorithm on different data sets. When comparing our scheme with k-median clustering [19], we used an enhanced version of the latter. The algorithm is allowed to keep a constant number of sample points per cluster, which can be used to deduce the approximate shape of that cluster. We ran the k-medians clustering using k clusters and total memory (number of samples) equal to m. CURE already has a parameter for maintaining samples in each cluster, so we used that feature. In this section, we analyze the output of these three algorithms (ClusterHull, k-median, CURE) on a variety of different data sets, and as a function of m, the memory.

Throughout this section, we use the *period-doubling cleanup* along with the area-perimeter cost (Equation 3.2) to compute the hulls. We use $\mu = .05$ and r, the number of initial sample directions per hull, equal to 8. The values of these parameters are critical for our algorithm; however, in this section *we use the same set of parameters for all data sets.* This shows that when tuned properly, our algorithm can generate good quality clusters for a variety of input distributions using a single set of parameters. In the next section, we will analyze in detail the effects of these parameters on the results of our scheme. To visualize the output, we also shade the hulls generated by our algorithm according to their densities (darker regions are more dense).

5.1 West Nile virus spread

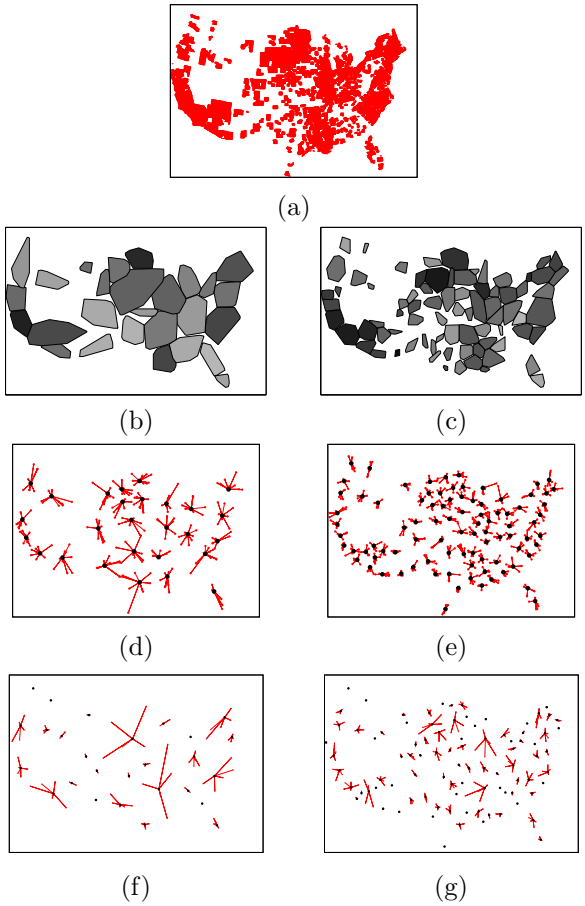


Figure 13: The *westnile* data set is shown in the top figure (a). Figures (b) and (c) show the outputs of ClusterHull for $m = 256$ and $m = 512$. Figures (d) and (e) show the corresponding outputs for k-medians. Figures (f) and (g) show the corresponding outputs for CURE.

Our first data set, *westnile* (Figure 13 (a)), contains about 68,000 points corresponding to the locations of the *West Nile* virus cases reported in the US, as collected by the CDC and the USGS [29]. We randomized the input order to eliminate any spatial coherence that might give an advantage to our algorithm. We ran ClusterHull to generate output of total size $m = 256$

and 512 (Figures (b) and (c)). The clustering algorithms k-medians and CURE were used to generate clusters with the same amount of memory. The results are shown in Figure 13.

All three algorithms are able to track high-density regions in coherent clusters, but there was little information about the shapes of the clusters in the output of k-medians or CURE. Visually the output of ClusterHull looks strikingly similar to the input set, offering the analyst a faithful yet compact representation of the geometric shapes of important regions.

5.2 The *circles* and the *ellipse* data sets

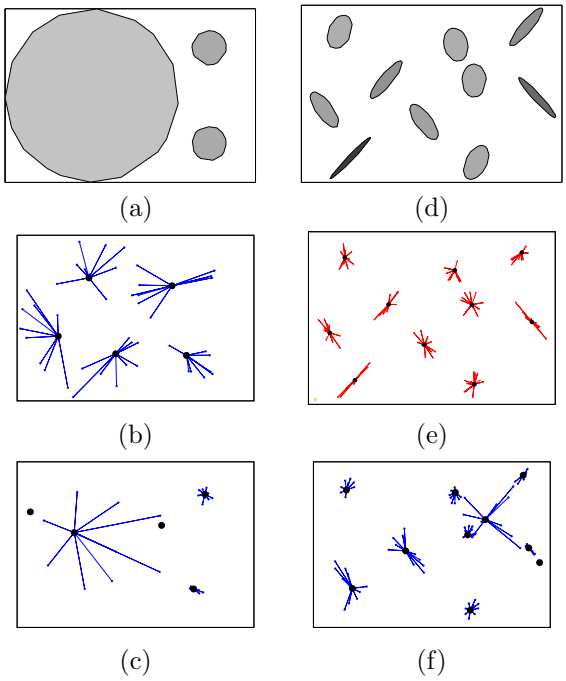


Figure 14: The left column shows output of ClusterHull (top), k-medians (middle) and CURE (bottom) for *circles* dataset with $m = 64$. The right column shows corresponding outputs for *ellipse* dataset with $m = 128$.

In this experiment, we compared ClusterHull with k-median and CURE on the *circles* and the *ellipse* data sets described earlier. The circles set contains $n = 10,000$ points generated inside 3 circles of different sizes. We ran the three algorithms with a total memory $m = 64$. The output of ClusterHull is shown in Figure 14 (a); the output of k-median is shown in (b); and the output of CURE is shown in (c).

Similarly, Figures 14 (d), (e), and (f), respectively, show the outputs of ClusterHull, k-median, and CURE on the ellipse data set with memory $m = 128$. The ellipse data set contains $n = 10,000$ points distributed among ten ellipse-shaped clusters.

In all cases, ClusterHull output is more accurate, visually informative, and able to compute the boundaries of clusters with remarkable precision. The outputs of other schemes are ambiguous, inaccurate, and lacking in details of the cluster shape boundary. For the circles data, the k-median does a poor job in determining the true cluster structure. For the ellipse data, CURE does a poor job in separating the clusters. (CURE needed a much larger memory—a "window size" of at least 500—to separate the clusters correctly.)

6 Tuning ClusterHull Parameters

In this section, we study the effects of various parameters on the quality of clusters.

6.1 Variation with r

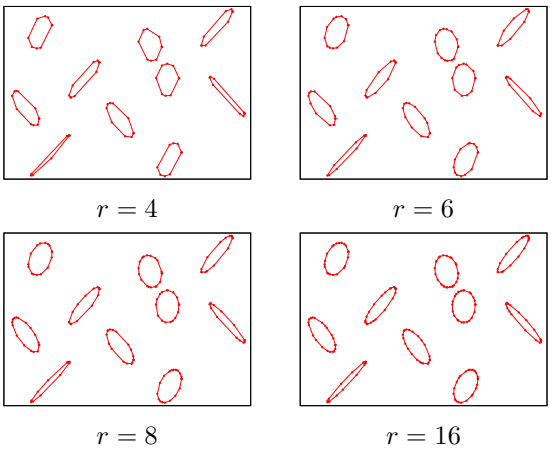


Figure 15: The result of varying r on the *ellipses* data set.

We first consider the effect of changing r, the number of initial directions assigned to each hull. To isolate the effects of r, we fixed the values of $\mu = .05$ and $k = 10$. We ran the experiments on two data sets, *ellipses* and *circles*. The results are shown in Figures 15 and 16, respectively.

The results show that the shape representation with 4 initial directions is very crude: ellipses are turned into pointy polygons. As we increase r, the representation of clusters becomes more refined. This contrast can be seen if we compare the boundary of

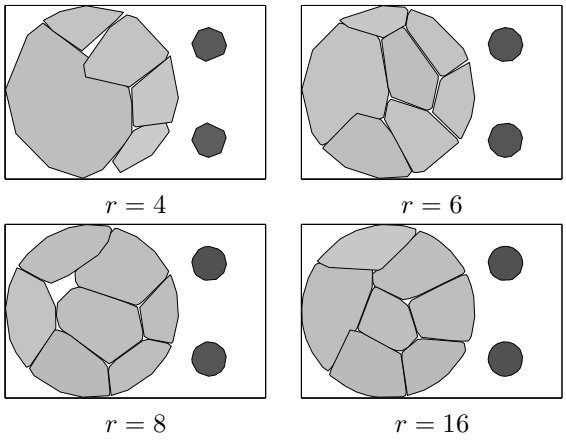


Figure 16: The result of varying r on the *circles* data set.

the big circle in Figure 16 for $r = 4$ and 8. However, increasing the number of directions means that we need more memory for the hulls (memory grows linearly with r). For $r = 8$, we get a good balance between memory usage and the quality of the shapes.

6.2 Variation with μ

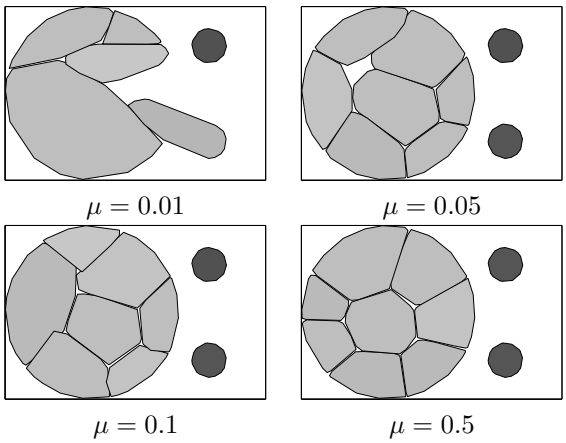


Figure 17: For the *circles* data set, ClusterHull recovers clusters correctly for $\mu \in [.05, .5]$, but fails for $\mu \leq .01$.

We considered values of μ in the range $[.01, .5]$. We fixed $r = 8$, and ran our algorithm for two data sets, *circles* and *stripes*. We fixed the number of hulls, $k = 10$ ($m = 128$) for *circles* and $k = 5$ ($m = 64$) for *stripes*.

If the value of μ is too small, the area dominates the cost function. This causes distant hulls to merge into long skinny hulls spanning multiple clusters. Although the period-doubling cleanup gets rid of most of them by discarding hulls with small densities, the output still contains some hulls spanning multiple natural clusters. Figure 17 shows this effect when $\mu = .01$.

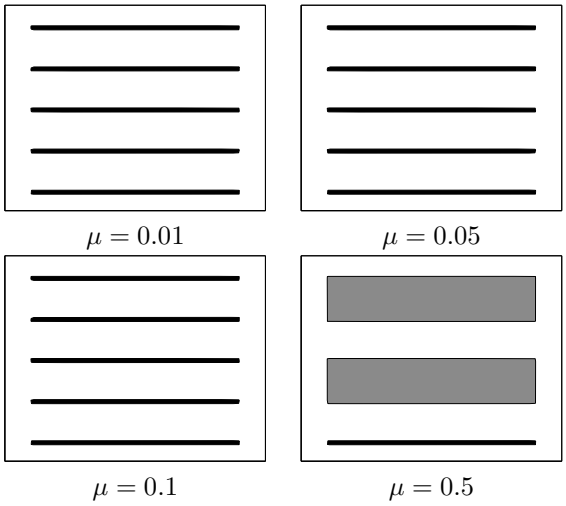


Figure 18: For the *stripes* data set, ClusterHull recovers clusters correctly for $\mu \in [.01, .1]$, but fails for $\mu \geq .5$.

On the other hand, if μ is increased too much, the cost function prefers decreasing the total perimeter, and it is hard to prevent large neighboring clusters from merging together. In Figure 18, neighboring stripes are merged into a single hull for $\mu = .5$. The results show that choosing μ in the range $[.05, .1]$ gives good clusters for most input sets.

7 Conclusion

We developed a novel framework for summarizing the geometric shape and distribution of a two-dimensional point stream. We also proposed an area-based quality measure for such a summary. Unlike existing stream-clustering methods, our scheme adaptively allocates its fixed memory budget to represent different clusters with different degrees of detail. Such an adaptive scheme can be particularly useful when the input has widely varying cluster structures, and the boundary shape, orientation, or volume of those clusters can be important clues in the analysis.

Our scheme uses a simple and natural cost function to control the cluster structure. Experiments show that this cost function performs well across widely different input distributions. The overall framework of ClusterHull is flexible and easily adapted to different applications. For instance, we show that the scheme can be enhanced with period-doubling and incremental

cleanup to deal effectively with noise and extreme data distributions. In those settings, especially when the input has spatial coherence, our scheme performs noticeably better than general-purpose clustering methods like CURE and k-medians.

Because our hulls tend to be more stable than, for instance, the centroids of k-medians, we can maintain other useful data statistics such as *population count* or *density* of individual hulls. (Our hulls grow by merging with other hulls, whereas the centroids in k-medians potentially shift after each new point arrival. The use of incremental cleanup may cause some of our hulls to be discarded, but that happens only for very low-weight, and hence less-interesting hulls.) Thus, the cluster hulls can capture some important frequency statistics, such as which five hulls have the most points, or which hulls have the highest densities, etc.

Although ClusterHull is inspired by the α-hull and built on top of an optimal convex hull structure, the theoretical guarantees of those structures do not extend to give approximation bounds for ClusterHull. Providing a theoretical justification for ClusterHull's practical performance is a challenge for future work.

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In Proc. of the 29th VLDB conference, 2003.

[3] N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* **58** (1999), 137–147.

[4] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60:125–135, 1998.

[5] P. S. Bradley, U. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. Proc. 4th International Conf. on Knowledge Discovery and Data Mining (KDD-98), 1998.

[6] T. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 152–159, 2004.

[7] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th Annu. ACM Symp. Theory Computing*, pages 626–635, 1997.

[8] D. Cheng, R. Kannan, S. Vempala, G. Wang A Divide-and-Merge Methodology for Clustering. In Proc. of the ACM Symposium on Principles of Database Systems, 2005.

[9] G. Cormode and S. Muthukrishnan. Radial histogram for spatial streams. Technical Report 2003-11, DIMACS, 2003.

[10] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH 96*, pages 303–312, 1996.

[11] T. K. Dey, K. Mehlhorn, and E. Ramos. Curve reconstruction: Connecting dots with good reason. *Comp. Geom.: Theory and Appl.*, 15:229–244, 2000.

[12] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In Proc. of the 18th International Conference on Machine Learning, ICML, 2001

[13] P. Domingos, G. Hulten. Learning from Infinite Data in Finite Time. In Proc. Advances in Neural Information Processing Systems (NIPS), 2002

[14] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In Proc. of 10th Symposium on Discrete Algorithms (SODA), 1999.

[15] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[16] M. Ester, H. Kriegel and X. Xu. A database interface for clustering in large spatial databases. In Int. Conference on Knowledge Discovery in Databases and Data Mining (KDD-95), Montreal, Cananda, August 1995.

[17] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models, 2002. Manuscript.

[18] A. Frieze, R. Kannan and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In Proc. of 39th Symposium on Foundations of Computer Science (FOCS), 1998.

[19] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. *Clustering data streams: Theory and practice.* IEEE Trans. Knowl. Data Engineering, Vol. 15, pages 515–528, 2003.

[20] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In Proc. of International Conference on Management of Data (SIGMOD), 1998.

[21] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. Proc. of SIGMOD, pages 58–66, 2001.

[22] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. In *Proc. 23rd ACM Sympos. Principles Database Syst.*, pages 252–262, 2004.

[23] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[24] S. Muthukrishnan. Data streams: Algorithms and applications. Preprint, 2003.

[25] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In Proceedings of the 20th VLDB Conference, 1994.

[26] J. O'Rourke and G. T. Toussaint. Pattern recognition. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 43, pages 797–813. CRC Press LLC, Boca Raton, FL, 1997.

[27] D. Pelleg, and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proc. of the 17th International Conference on Machine Learning (ICML), 2000.

[28] S. Vempala R. Kannan and A. Vetta On clusterings - good, bad and spectral. In Proc. 41st Symposium on the Foundation of Computer Science, FOCS, 2000.

[29] USGS West Nile Virus Maps - 2002. `http://cindi.usgs.gov/hazard/event/west_nile/`.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In Proc. of the International Conference on Management of Data (SIGMOD), 1996.