Fast Computation of 2-Dimensional Depth Contours

Ted Johnson
AT&T Research Center
Florham Park, NJ.
Email: johnsont@research.att.com

Ivy Kwok and **Raymond Ng** *
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4 Canada
Email: {ikwok,rng}@cs.ubc.ca

Abstract

"One person's noise is another person's signal." For many applications, including the detection of credit card frauds and the monitoring of criminal activities in electronic commerce, an important knowledge discovery problem is the detection of exceptional/outlying events.

In computational statistics, a depth-based approach detects outlying data points in a 2-D dataset by, based on some definition of depth, organizing the data points in layers, with the expectation that shallow layers are more likely to contain outlying points than are the deep layers. One robust notion of depth, called depth contours, was introduced by Tukey. ISODEPTH, developed by Ruts and Rousseeuw, is an algorithm that computes 2-D depth contours.

In this paper, we give a fast algorithm, FDC, which computes the first k 2-D depth contours by restricting the computation to a small selected subset of data points, instead of examining all data points. Consequently, FDC scales up much better than ISODEPTH. Also, while ISODEPTH relies on the non-existence of collinear points, FDC is robust against collinear points.

Keywords: depth contours, computational statistics, convex hulls

Introduction

Knowledge discovery tasks fall into four general categories: (a) dependency detection, (b) class identification, (c) class description, and (d) exception/outlier detection. The first three categories correspond to patterns that apply to many, or a large percentage of, objects in the dataset. The fourth category, in contrast, focuses on a very small percentage of data objects, which is often ignored or discarded as noise. Some existing algorithms in machine learning and data mining have considered outliers, but only to the extent of tolerating them in whatever the algorithms are supposed to do (Angluin & Laird 1988; Ester *et al.* 1996; Ng & Han 1994; Zhang, Ramakrishnan, & Livny 1996). However, "one person's noise is another person's signal." Indeed, for some applications, the rare events are often more interesting than the common ones, from a knowledge discovery standpoint. Sample applications include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

Most of the existing works on outlier detection lies in the field of statistics (Barnett & Lewis 1994; Hawkins 1980). While there is no single, generally accepted, formal definition of an outlier, Hawkins' definition captures the spirit: "an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" (Hawkins 1980). Accordingly, over one hundred discordancy/outlier tests have been developed for different circumstances, depending on: (i) the data distribution, (ii) whether or not the distribution parameters (e.g., mean and variance) are known, (iii) the number of expected outliers, and even (iv) the types of expected outliers (e.g., upper or lower outliers in an ordered sample) (Barnett & Lewis 1994). However, all of those tests suffer from the following two problems. First, most of them are univariate (i.e., single attribute). This restriction makes them unsuitable for multidimensional datasets. Second, all of them are distribution-based. In numerous situations, we do not know the data distribution and have to perform extensive testing to find a distribution that fits the attribute. Although a unified outlier detection system (Knorr & Ng 1997) can handle situations where the attribute follows any distribution, like any distance-based data mining works, the detection requires the existence of a metric distance function, which is not available for every dataset. In (Arning, Agrawal, & Raghavan 1996), Arning *et al.* search a dataset for implicit redundancies, and extract data objects called *sequential exceptions* that maximize the reduction in Kolmogorov complexity. This notion of outliers is very different from the aforementioned statistical definitions of outliers. As will be seen shortly, it is also very different from the notion of outliers considered here, primarily because there is not an associated notion of depth.

To avoid the aforementioned problems of distribu-

* Person handling correspondence.

tion fitting and restriction to univariate datasets, *depth-based* approaches have been developed. In these approaches, each data point is assigned a depth. Based on the assigned depth, data objects are organized in layers in the data space, with the expectation that shallow layers are more likely to contain outlying data objects than are the deep layers. A key property of depth-based approaches is that location depth is scaling-invariant

A robust notion of depth, called *depth contour* was introduced by Tukey (Tukey 1975; 1977). Intuitively, a point P in space is of depth k if k is the minimum number of data points that have to be removed to expose P. Here "minimum" is defined across all the half planes passing through P. The k-th depth contour marks the boundary between all points with depth k and all those with depth $k+1$. Figure 1(a) shows the first 20 depth contours for a dataset containing 5,000 points. In general, depth contour maps gives a nice picture of the "density" of the outlying regions (Liu, Parelius, & Singh 1997). In (Ruts & Rousseeuw 1996), Ruts and Rousseeuw develop an algorithm called ISODEPTH, which computes 2-D depth contours.

This paper extends the work of Ruts and Rousseeuw in two key areas. First, ISODEPTH relies on the computation of *dividers* (to be formally introduced later) for all n data points in the dataset. Having a complexity at least quadratic in n, ISODEPTH does not scale up well. In contrast, to compute the first k depth contours, our algorithm FDC restricts the computation to a selected, much smaller, subset of points, thanks to the construction of the appropriate convex hulls. Consequently, FDC can scale up much better than ISODEPTH. Second, ISODEPTH relies on the non-existence of collinear points. Removing all collinear points can be very time consuming. FDC is robust against collinear points. Figure 1(b) shows the depth contours computed by FDC for concentric data points, many of which are collinear.

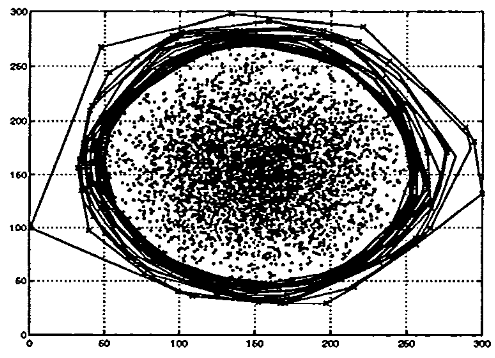
Algorithm FDC

Definition 1 Given a point cloud D consisting of n points, a line L is an e-divider of D if there are e points in D to be left of L, and $(n-e)$ points in D to the right of L.

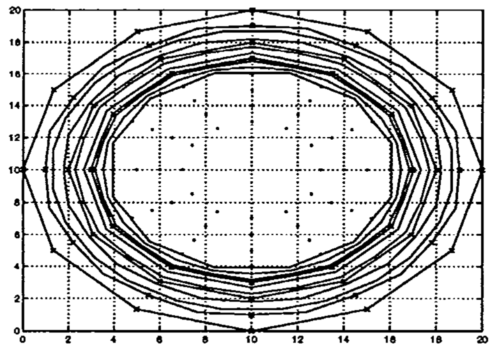
In the spirit of finding outliers, whenever $e \leq (n-e)$, we say that the e points are to the "outside" of L and the remaining points to the "inside" of L. Just as an e-divider L divides the data cloud D into two disjoint subsets, it divides the convex hull of D into two subregions. We call these the "inside region" and the "outside region," denoted as $IR(L)$ and $OR(L)$ respectively. Given a collection of e-dividers, we refer to the intersection of all their inside regions (i.e., $\bigcap IR(L)$) as the *e-intersected inside region.*

As we shall see later, the only interesting part of an e-divider is a finite segment of it. We denote a line segment between points P and Q by $\langle P, Q\rangle$. The above definition of e-dividers can be extended to line segments. More specifically, the depth of a line segment is the depth of the line that runs through the line segment. A chain of line segments, denoted as $\langle P_1, P_2, \ldots, P_n\rangle$ (where $P_i \neq P_j$ for all $i \leq j$), is the set of line segments $\langle P_i, P_{i+1}\rangle$ for $1 \leq i \leq (n-1)$. For example, $IR(\langle P_1, P_2, \ldots, P_n\rangle)$ denotes the inside region of the convex hull of D, i.e., the region of all the points in the convex hull that are to the inside of the chain. As will be obvious later, every line segment in the chain is an e-divider. We call the chain an *expanded e-divider.* Given any number of e-dividers but at least one expanded e-divider, we refer to the intersection of all their inside regions as the *expanded* e-intersected inside region.

Figure 2 shows the pseudo code of Algorithm FDC (to stand for "Fast Depth Contour"), which computes the first k depth contours of a bivariate point cloud. While we shall prove the correctness of the algorithm shortly, the following first gives an example to show how the algorithm works.



(a) 5,000 Data Points



(b) Concentric Data Points

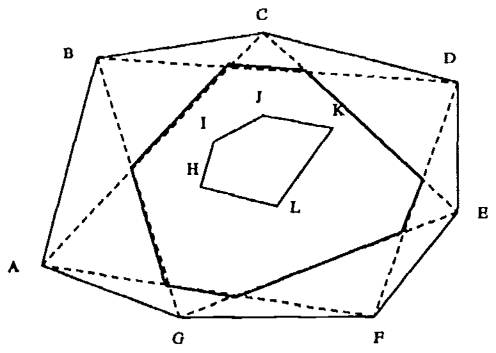
Figure 1: Depth Contours

An Example

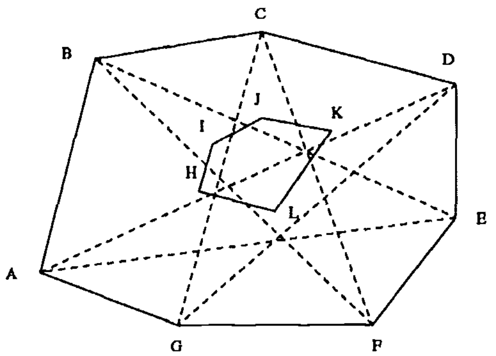
Consider the situation depicted in Figure 3. Suppose that the point cloud contains many points, among which only a few are shown. In particular, suppose that

```
Algorithm FDC
Input: $D \equiv$ the point cloud; $k \equiv$ an integer.
Output: Contours of depth from 0 to $k$.
1 $G =$ convex hull of $D$;
2 For ($d = 0$; $d < k$; ) { /* new peel */
  2.1 $H = G$;
  2.2 Output $H$ as the $d$-th depth contour;
  2.3 If $d == k$, break; /* Done and Stop */
  2.4 $d = d + 1$; $D = D - H$; $G =$ convex hull of (the updated) $D$;
  2.5 If $|H| == 3$, continue; /* degenerate case of having a triangle as the
      convex hull */
  2.6 For ($e = 1$; $e < |H|/2$; ) { /* Otherwise: the general case */
   2.6.1 For all points $P \in H$ {
    2.6.1.1 Find the point $P_e \in H$ (if any) so that $\langle P, P_e \rangle$ is an $e$-divider of
            the remaining points in $H$.
    2.6.1.2 If every point in $G$ is contained in $IR(\langle P, P_e \rangle)$ {
    2.6.1.2.1 $IR[P] = IR(\langle P, P_e \rangle)$; $AH[P] = \emptyset$; } /* end if */
    2.6.1.3 Else { /* form an expanded e-divider */
    2.6.1.3.1 Enumerate (in a clockwise fashion) all the points in $G$ that are
              outside $\langle P, P_e \rangle$ as $Q_1, \ldots, Q_m$.
    2.6.1.3.2 Among them, find $Q_i$ that maximizes the angle between the
              segment $\langle Q_i, P \rangle$ and $\langle P, P_e \rangle$.
    2.6.1.3.3 Among them, find $Q_j$ that maximizes the angle between the
              segment $\langle Q_j, P_e \rangle$ and $\langle P_e, P \rangle$. /* Without loss of generality,
              assume that $i \leq j$. */
    2.6.1.3.4 $IR[P] = IR(\langle P, Q_i, \ldots, Q_j, P_e \rangle)$; $AH[P] = \{Q_i, \ldots, Q_j\}$; } /*
              end else */
          } /* end for */
   2.6.2 Output the boundary of the region $(\bigcap_{P \in H} IR[P])$ as the $d$-th
         depth contour;
   2.6.3 If $d == k$, break; /* Done and Stop */
   2.6.4 /* Otherwise */ $d = d + 1$; $e = e + 1$;
   2.6.5 If $\bigcup_{P \in H} AH[P] \neq \emptyset$ { /* G is not contained in the e-intersected
         inside region $\bigcap_{P \in H} IR(\langle P, P_e \rangle)$ */
   2.6.5.1 $H = H \cup (\bigcup_{P \in H} AH[P])$; $D = D - (\bigcup_{P \in H} AH[P])$;
   2.6.5.2 $G =$ convex hull of (the updated) $D$; } /* end if */
      } /* end for */
  } /* end for */
```

Figure 2: Pseudo Code of Algorithm FDC



(a) 1-Dividers



(b) 2-Dividers

Figure 3: An Example: 1-Dividers and 2-Dividers

in Step 1 of Algorithm FDC, the convex hull of the entire point cloud is found to be the polygon with vertices from A to G. In the first iteration of the for-loop in Step 2, this polygon is returned as the zero-th depth contour in Step 2.2.

Suppose that in Step 2.4, the convex hull of the remaining points in the data cloud (i.e., all points except from A to G) is found to be the polygon with vertices from H to L. Then in the first iteration of the for-loop in Step 2.6, all the 1-dividers are found. For instance, let us consider one case, say A. The 1-divider of A is $\langle A, C \rangle$. The inside region $IR(\langle A, C \rangle)$ is precisely the polygon with vertices A, C, D, E, F, G. This polygon completely contains the polygon with vertices from H to L. Thus, Step 2.6.1.2 is executed, but Step 2.6.1.3 is avoided. In fact, this is the case for every point from A to G and its corresponding 1-divider. Therefore the region $(\bigcap_{P \in H} IR[P])$ computed in Step 2.6.2 is the 1-intersected inside region formed by all the 1-dividers, marked explicitly in Figure 3(a). Every point X inside this region has at least two points from A to G that are outside of any line passing through X. And for any point X outside the region (but inside the polygon with vertices from A to G), there exists at least one line passing through X that separates exactly one point among A to G from the rest. The boundary of this 1-intersected inside region gives the contour of depth equal to 1.

Now in the second iteration of the for-loop in Step 2.6, the 2-dividers are considered. As shown in Figure 3(b), this time it is no longer the case that every inside region completely contains the polygon with vertices from H to L. In other words, the 2-intersected inside region does not completely contain the latter polygon. As a concrete example, consider A. As shown in Figure 4, the 2-divider is $\langle A, D \rangle$, and the points H, I, J and K are to the outside of $\langle A, D \rangle$. Thus, in Step 2.6.1.3.2 the point that maximize the angle formed by the point, A and D is computed. In this case, I is the point (i.e., the angle formed by I, A and D is bigger than that formed by H, A and D, and so on). Similarly, in Step 2.6.1.3.3, it is found that J maximizes the angle formed by the point, D and A. Thus, the original 2-divider $\langle A, D \rangle$ is expanded and replaced by the chain $\langle A, I, J, D \rangle$. Accordingly, the original inside region with vertices A, D, E, F, G is expanded to become the polygon with vertices A, I, J, D, E, F, G. Eventually, in Step 2.6.2, the expanded divider $\langle A, I, J, D \rangle$ defines part of

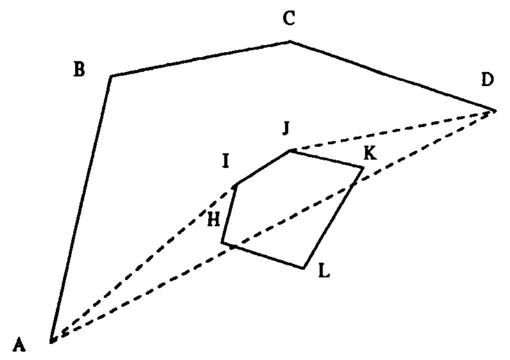


Figure 4: An Example (cont'd): Expanding a 2-Divider

the boundary of the contour of depth equal to 2.

To understand why $\langle A, D\rangle$ should be expanded by $\langle A, I, J, D\rangle$ in forming part of the boundary of the contour of depth 2, it is easy to verify from Figure 4 that apart from B and C, there are at least the additional points H to K that are outside of $\langle A, D\rangle$. In contrast, the chain $\langle A, I, J, D\rangle$ ensures that for any point X in the polygon with vertices A, B, C, D, J and I (and inside the contour of depth 1), there exists at least one line passing through X that separates B and C from the rest.

Note that in Step 2.6.5, I and J are added to the the set of points among which e-dividers are to be found in subsequent iterations of the for-loop in Step 2.6. These two points are added because of $\langle A, D\rangle$. To complete the example shown in Figure 3(b), point H is also added because H expands the 2-divider $\langle F, B\rangle$. Similarly, point K is added because K expands both $\langle B, E\rangle$ and $\langle C, F\rangle$.

So far the example shown in Figure 3 has illustrated the major parts of the algorithm. But one aspect that requires further explanation is the case being dealt with in Step 2.5. Assume that there are 3 extra points X, Y, Z that form a triangle encompassing the polygon with vertices from A to G in Figure 3. In this case, the triangle gives the first convex hull of the dataset, and is therefore the contour of depth 0. Then it is easy to verify that the next convex hull – namely, the polygon with vertices from A to G – is the contour of depth 1. This illustrates why in Step 2.5, if the convex hull is a triangle, the algorithm can simply proceed to the remaining points.

Correctness and Analysis of FDC

Consider every instance of the set H as Algorithm FDC executes. Each instance of H can be labeled as $H_{i,e}$ indicating the content of the set at the *beginning* of the e-th iteration of the for-loop in Step 2.6, but during the i-th iteration of the for-loop in Step 2. Under this labeling scheme, the set H before entering Step 2.6 (i.e., between Steps 2.1 and 2.5) is represented by $H_{i,1}$. Thus, as Algorithm FDC executes, it produces the series $H_{1,1}, \ldots, H_{1,m_1}, H_{2,1}, \ldots, H_{2,m_2}, H_{3,1}, \ldots$. A key result is that there is a one-to-one correspondence between the series $H_{1,1}, \ldots, H_{1,m_1}, H_{2,1}, \ldots, H_{2,m_2}, H_{3,1}, \ldots$ and the contours of depth 0, 1, 2, ... In particular, the list of contours of depth 0, 1, 2, ... is the list:

$$H_{1,1}, \bigcap_{P \in H_{1,2}} IR[P], \bigcap_{P \in H_{1,3}} IR[P], \ldots, \bigcap_{P \in H_{1,m_1}} IR[P],$$
$$H_{2,1}, \bigcap_{P \in H_{2,2}} IR[P], \bigcap_{P \in H_{2,3}} IR[P], \ldots, \bigcap_{P \in H_{2,m_2}} IR[P],$$
$$\ldots$$

For more detail, please read (Johnson, Kwok, & Ng 1998).

Now we consider complexity/efficiency issue. Let n be the total number points in the data cloud. Let h denote the maximum cardinality of the first k elements in the series $H_{1,1}, \ldots, H_{1,m_1}, H_{2,1}, \ldots, H_{2,m_2}, \ldots$. Similarly, let g denote the maximum cardinality of the first k instances G. A complexity analysis of FDC is deferred to a more detailed report /citeNg98. The convex hull computation and maintenance takes $O(nlogn+h\ log\ ^2n)$ and the rest of computation, with Step 2.6.1.1 (finding the initial e-dividers) dominating other steps, takes $O(kh^3)$. This gives an overall complexity of $O(n\ logn\ + h\ log\ ^2n\ +\ kh^3)$.

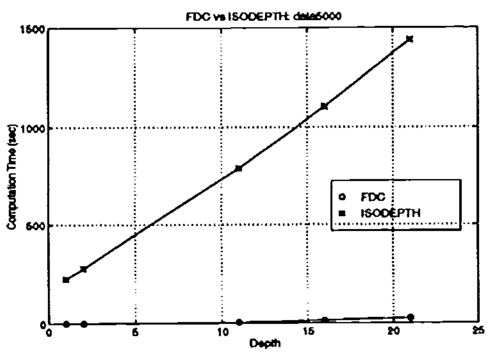
How does the complexity of FDC compare with ISODEPTH's $O(n^2 logn)$? This comparison boils down to the relative magnitudes of h, k and n. From the point of view of finding outliers in large datasets, k is typically not large (say, ≤ 100, if not smaller) and h (which is also partly dependent on k) is at least 2-3 orders of magnitude smaller than n. Thus, for the intended uses of the algorithms, we expect FDC to outperform ISODEPTH when n is not too small.

Preliminary Experimental Results

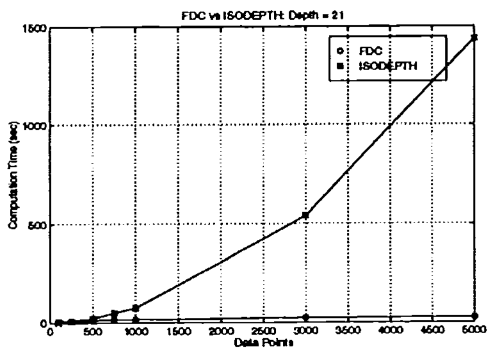
Below we present experimental results comparing ISODEPTH with FDC. We obtained a copy of ISODEPTH from Ruts and Rousseeuw; the program was written in Fortran. We implemented FDC in C++, and used the LEDA library for most computational geometry operations, such as convex hull computations and polygon intersections. As for datasets, we used both real and generated ones. All graphs shown here are based on generated datasets; but the conclusions we draw generalize to many real datasets.

Figure 5(a) shows the computation time taken by the two algorithms to produce depth contours from 0 to k, with k varying between 1 and 21. The computation time consists of the CPU time taken to produce all k depth contours after the data points are loaded. Figure 5(a) is based on the dataset consisting of 5,000 points shown in Figure 1(a). It is clear that FDC outperforms ISODEPTH by at least an order of magnitude. For example, for $k = 21$, FDC takes about 25 seconds, whereas ISODEPTH takes at least 1,400 seconds.

Figure 5(b) shows how the two algorithms scale up with respect to the size of the dataset, with k



(a) Wrt the Number of Depth Contours



(b) Wrt the Dataset Size

Figure 5: FDC vs ISODEPTH

set to 21. When there are fewer than 500 points, the two algorithms are competitive. But with 1,000 points or more, FDC scales up much more nicely than ISODEPTH does. For instance, FDC is 4 times faster than ISODEPTH for 1,000 points, but is more than 50 times faster for 5,000 points.

The version of ISODEPTH we have cannot run when $n \geq 5,000$. This is due to the excessive amount of space used by the program. This is why our head-to-head comparisons between the two algorithms stop at 5,000. The table below shows that FDC is very scalable with respect to n. Although the figures are based on a constant k (depth = 21), FDC also seems to scale up well with respect to k in Figure 5.

Dataset Size (n)	1,000	10,000	100,000
Maximum Cardinality (h)	131	174	199
Computation Time (sec)	17	27	52

Future Work

In the above analysis, we show a kh^3 factor in the complexity figure. Even though h is supposedly very small compared with n and the factor kh^3 poses no problem in the many cases we have experimented with, it is conceivable that if k is a large value, then h will be large as well. As factor kh^3 corresponds to the finding e-dividers, we are investigating how to further optimize this step.

There is also no easy way to formulate a pattern for the k values as each dataset has its own characteristics. One approach is to make use of the value h. We can specify the number or percentage of data objects we want, and then compute the depth contour till h meets the required value.

Lastly, we are working on generalizing the 2-dimensioanl FDC to 3-dimension. We focus only on the 3-D case because the geometry computation in high dimensions is costly and the results are hard to be visuallized.

References

Angluin, D., and Laird, P. 1988. Learning from noisy examples. *Machine Learning* 2(4):343–370.

Arning, A.; Agrawal, R.; and Raghavan, P. 1996. A linear method for deviation detection in large databases. In *Proc. KDD*, 164–169.

Barnett, V., and Lewis, T. 1994. *Outliers in Statistical Data*. John Wiley & Sons.

Ester, M.; Kriegel, H.-P.; Sander, J.; and Xu, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD*, 226–231.

Hawkins, D. 1980. *Identification of Outliers*. London: Chapman and Hall.

Johnson, T.; Kwok, I.; and Ng, R. T. 1998. FDC: Fast computation of 2-dimensional depth contours. Unpublished Manuscript, Dept. of Computer Science, University of British Columbia.

Knorr, E. M., and Ng, R. T. 1997. A unified notion of outliers: Properties and computation. In *Proc. KDD*, 219–222.

Liu, R.; Parelius, J.; and Singh, K. 1997. Multivariate analysis by data depth: Descriptive statistics, graphics, inference. In *Technical Report, Dept. of Statistics, Rutgers University*.

Ng, R., and Han, J. 1994. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th VLDB*, 144–155.

Ruts, I., and Rousseeuw, P. 1996. Computing depth contours of bivariate point clouds. *Computational Statistics and Data Analysis* 23:153–168.

Tukey, J. 1975. Mathmatics and the picturingof data. In *Proc. International Congress on Mathmatics*, 523–531.

Tukey, J. 1977. *Exploratory Data Analysis*. Addison-Wesley.

Zhang, T.; Ramakrishnan, R.; and Livny, M. 1996. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD*, 103–114.