Fast anomaly discovery given duplicates

Jay-Yoon Lee, U Kang, Danai Koutra,
Christos Faloutsos

Dec 2012
CMU-CS-12-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Given a large cloud of multi-dimensional points, and an off-the-shelf outlier detection method, why does it take a week to finish? After careful analysis, we discovered that duplicate points create subtle issues, that the literature has ignored: if d_{max} is the multiplicity of the most over-plotted point, typical algorithms are quadratic on d_{max}. For graph-related outlier detection, all the satellites of a 'star' node will have identical features, and thus create over-plotting with d_{max} being the highest degree; due to power law degree distributions, this may be huge, for real graph data. We propose several ways to eliminate the problem; we report wall-clock times and our time savings; and we show that our methods give either exact results, or highly accurate approximate ones.

This research was sponsored by the United States Army under grant number W1911NF-11-C-0088. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: computer science, tech reports, anomaly detection, large graph

1 Introduction

Outlier detection, also known as anomaly detection, is an important area of data mining which has been receiving a lot of research attention [6, 23, 19, 9, 2] due to its numerous applications: fraud detection, health care, computer security, social network, etc.

The size of the data to apply outlier detection algorithms is growing at an unprecedented rate. For example, Facebook loads 10 Terabyte of new data every day [24]; Microsoft has click-through data reaching Petabyte scale [16]; Yahoo's web graph has 1.4 billion nodes and 6.6 billion edges [11]. These big data pose new problems, such as the "duplicate data point" problem, which refers to the existence of many data points with same coordinates. For example, assume a 2-D dataset <degree, PageRank> of nodes in a graph. Although for up to medium sized graphs duplicates are not an issue, in large billion-node graphs multiple nodes have the same <degree, PageRank> pairs.

Traditional outlier detection algorithms did not consider the duplicate data point problem for two reasons: (i) they dealt with relatively small amount of data with few -if any- duplicates, and (ii) most outlier detection algorithms work on Geographical Information System (GIS) data which do not have many duplicates as buildings, obviously, never exist over a building.

Challenges. This existence of the duplicate data points in traditional algorithms poses two challenges.

1. **Degeneracy.** The duplicate data points distort the existing algorithms to have degenerate results: (a) they act as black holes with extremely high densities and, thus, the normal-looking points near them are assigned as outliers, and (b) the outlier score may not be defined at all (Section 2.1).
2. **Running Time.** The duplicate points retard the computation of the existing algorithms; e.g., the running time grows from near-linear to near-quadratic (Section 2.2).

Our Contributions. In this paper we address the above challenges of outlier detection algorithms. Specifically, we focus on Local Outlier Factor (LOF), a powerful, widely-used outlier detection algorithm and propose optimizations. Our main contributions are:

1. **No Degeneracy.** We re-design the existing outlier detection algorithm to handle duplicate points and remove degeneracy.
2. **Running Time.** Our algorithms, FADD and G-FADD, run in near-linear time compared to the near-quadratic time of the existing algorithm.
3. **Discovery.** We run experiments on many real, large data and present interesting findings including Twitter accounts for advertisement spams, and nodes with high PageRanks despite small degrees.

The rest of the paper is organized in a typical way from background to proposed method to experiments and conclusions. Table 1 shows the symbols used in the paper.

Symbol	Definition
x_i	Multi-dimensional point.
X	Set of multi-dimensional points: $X = \{x_1, ..., x_N\}$.
N	The cardinality of X.
U	Set containing all unique elements of X: $U = \{u_1, ..., u_M\}$.
M	The cardinality of U.
c_i	Count of duplicates of a unique element u_i.
$SN(u_i)$	Super node: set of x_j that have the same coordinates as u_i; i.e. $\|SN(u_i)\| = c_i$.
k	Number of nearest neighbors in LOF.
p_i	Point of interest.
o_{ij}	Point in the k-distance neighborhood of p_i.
$d(p, o)$	Distance between points p and o.
$lof_k(x_i)$	Local outlier score of point x_i.
$lrd_k(x_i)$	Local density of point x_i.
l	Number of grids used in each dimension for G-FADD.

Table 1: Table of symbols and definitions.

2 Background and Observations

To address the "duplicate data point" issue, we focus on the Local Outlier Factor algorithm (LOF) [6] for two reasons: (a) the techniques that suffer most from the duplicates are based on k-nearest neighbor (*kNN*) method, and, as we will see next, LOF adopts this approach, and (b) it is the most widely used outlier detection scheme.

In a nutshell, LOF compares the local density of each data point to the densities of its neighbors by employing the *kNN* technique. Data points whose densities differ *much* from their neighbors' densities are flagged as outliers.

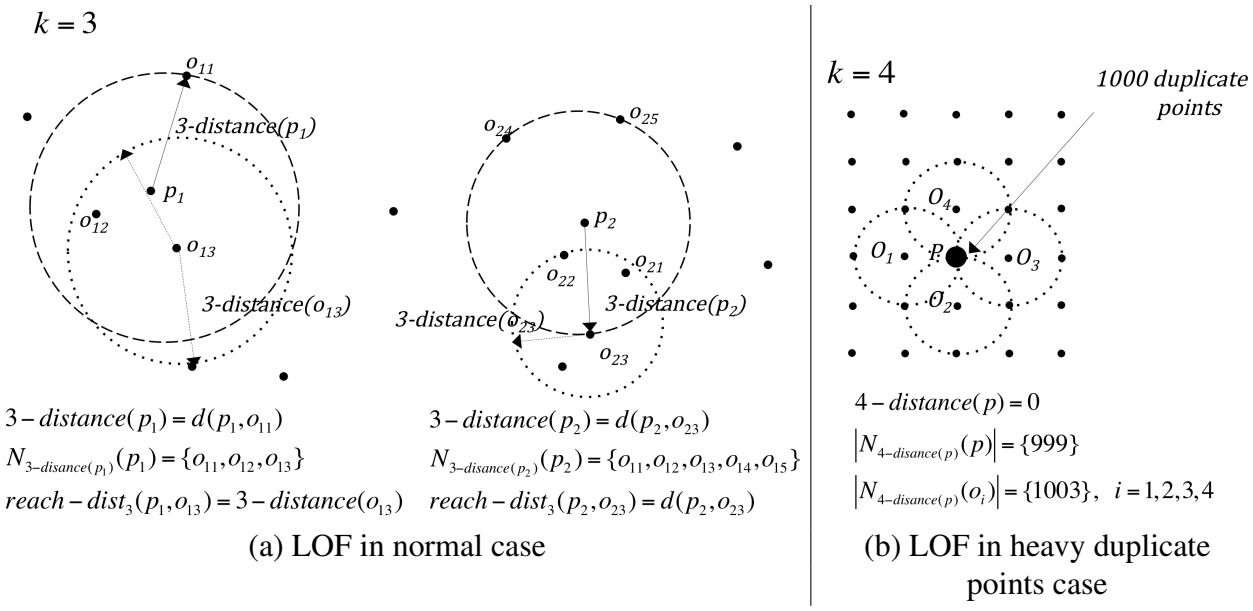
2.1 Definitions

Definition 1 (k-distance) *The k-distance of a data point p, k-distance(p), is defined for any* $k \in \mathbb{N}^+$ *as the distance* $d(p, o)$ *between the points* p *and* $o \in X$ *such that:*
(i) for at least k *objects* $o' \in X \setminus \{p\}$ *it holds that* $d(p, o') \leq d(p, o)$*, and*
(ii) for at most $k - 1$ *objects* $o' \in X \setminus \{p\}$ *it holds that* $d(p, o') < d(p, o)$. ■

For example, Figure 1 (a) shows that the 3-$distance$ of data point p_1 is the distance between p_1 and o_{11}, which is the distance between p_1 and its third nearest neighbor.

Definition 2 (k-distance neighborhood) *The k-distance neighborhood of an object* p *is given by* $N_k(p) = \{o \in X \setminus \{p\} \mid d(p, o) \leq k\text{-}distance(p)\}$. ■

In Figure 1 (a), $N_3(p_1) = \{o_{11}, o_{12}, o_{13}\}$, and $N_3(p_2) = \{o_{21}, o_{22}, o_{23}, o_{24}, o_{25}\}$. Note that $|N_3(p_2)| = 5 > 3$ because there are three ties between the distances $d(p_2, o_{23})$, $d(p_2, o_{24})$, and $d(p_2, o_{25})$. In Figure 1 (b), there are 1000 duplicate points on the position of p. What would be the



(a) LOF in normal case

(b) LOF in heavy duplicate points case

Figure 1: Illustration of concepts of $k-distance$, $N_k(p)$ and $reach\text{-}dist_k$ required for computation of LOF.

$4\text{-}distance$ neighborhood of p? It turns out that the size of the set is $|N_4(p)| = 999$. Furthermore, the neighboring points o_1, o_2, o_3, and o_4 will have $4\text{-}distance$ neighborhood of size 1003, since all the duplicate points in p will be counted as members of their $4\text{-}distance$ neighborhood. This phenomenon affects the runtime of the algorithm, as we present next.

Definition 3 (Reachability distance) *The reachability distance [6] of an object p w.r.t. object o is given by* $reach\text{-}dist_k(p,o) = max\{k\text{-}distance(o), d(p,o)\}$. ■

For example, in Figure 1 (a), $reach\text{-}dist_3(p_1, o_{13}) = 3\text{-}dsitance(o_{13})$, and $reach\text{-}dist_3(p_2, o_{23}) = d(p_2, o_{23})$. Conceptually, one can think this distance simply as physical distance between p and o.

Definition 4 (Local reachability density) *The local reachability density (*lrd*) of an object p is defined as*

$$lrd_k(p) = 1/(\frac{\sum_{o \in N_k(p)} reach\text{-}dist_k(p,o)}{|N_k(p)|}). \quad (1)$$

■

The local reachability density $lrd_k(p)$ is a measure of density w.r.t. distance. If we simply think of $reach\text{-}dist_k(p,o)$ as $d(p,o)$, then the denominator is just the average distance of $N_k(p)$ from p. In Figure 1 (b), the average distance in $N_4(p)$ is 0, because all the neighborhoods have the same coordinates.The ill-defined *lrd* in the case of duplicate points is an additional problem to the large neighborhood size that we mentioned above.

Definition 5 (Local outlier factor) *The local outlier factor of an object p is:*

$$LOF_k(p) = |N_k(p)|^{-1} \sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)} \quad (2)$$

■

Based on Equation (2), a point p is outlier if its *lrd* is relatively small compared to its neighborhoods' *lrd*'s. That is, a point whose density is *different* from the densities of its neighbors is likely to be an outlier. Note that the LOF score of a point with more than k duplicates should be 1, since the point and all its k neighbors are identical.

2.2 Observations: LOF Runtime and Duplicate Points

We show an example illustrating the problem caused by duplicate points. Consider the case of Figure 1 (b): $LOF_k(p)$ calls $lrd_k(p)$ 999 times, and each $lrd_k(p)$ calls 999 $reach\text{-}dist_k(p, o)$ since each runs *kNN* once. Thus, *kNN* is called in total 998,001 times. Although data structures that use *kNN* algorithms mostly segment the coordinate space and do not search the whole space for every point, duplicates cannot be separated; for example, in Figure 1 (b), the 999 points cannot be separated.

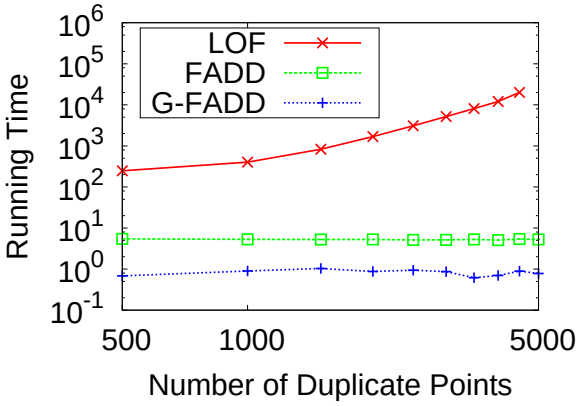


Figure 2: Running time of LOF, FADD, and G-FADD on 50k uniformly distributed points with different number of duplicate points. LOF suffers from the quadratic complexity while FADD and G-FADD are less sensitive to the duplicates.

Observation 1 (LOF with duplicates) *LOF works fine with normal use, but has problems with large duplicate points because:*

1. *Large duplicates impede the calculation of LOF score by increasing the number of data access to $O(max({c_i}^2))$, where c_i is count of duplicates for unique element u_i, from $O(N)$ in no duplicate situation. The proof is omitted for brevity but the trend is illustrated in Figure 2.*
2. *$LOF_k(p)$ value is not well defined due to division by 0 in computation of local reachability density $lrd_k(p)$.*

2.3 Patterns of Duplicates in real data

How many duplicate points do real world data have? We show the patterns of duplicates from two real world data: Stack Overflow and US Patent (see Section 4) which have 244 K and 2 million points, respectively. We count the number of duplicate points and show the number of top 5 largest duplicates in Table 2. The top 5 duplicate counts comprise 6.70% and 12.90% of total number of points in Stack Overflow and US patent data, respectively. Also, they occupy 61.7% and 79.5% of the total sum of $count^2$, so more than half of the computation work described in Section 2.2 is spent on these points. To solve this problem of duplicates, we propose two fast outlier detection algorithms, FADD and G-FADD, in Section 4.

	Stack Overflow		US Patent	
Top5	*count*	*count*^{2}	*count*	*count*^{2}
1	4221	17.8 M	60598	3.7 B
2	3799	14.4 M	59744	3.6 B
3	3147	9.9 M	56191	3.2 B
4	2844	8.1 M	49192	2.4 B
5	2374	5.6 M	41929	1.8 B
sum	16385 (6.70%)	55.8 M (61.7%)	267654 (12.9%)	14.7 B (79.5%)

Table 2: (M: million, B: billion.) The number of points (*count*) and the number of points squared (*count*^{2}) for the top 5 largest duplicates in real world data. The top 5 duplicate counts comprise 6.70% and 12.90% of total number of points in Stack Overflow and US patent data, respectively. Also, they occupy 61.7% and 79.5% of the total sum of $count^2$, so more than half of the computation work described in Section 2.2 is spent on these points.

3 Proposed Method: Fast Outlier Detection

The challenges discussed in the previous section mainly arise from regarding as unique the points that are similar in the projected feature space, and handling them separately (e.g., all the <pagerank, degree> points with the same coordinates are viewed as distinct points). This approach causes a quadratic runtime complexity and renders outlier scores undefined. In this section, we present FADD, Fast Anomaly Detection algorithm given Duplicates, and G-FADD (Grid Fast Anomaly Detection algorithm given Duplicates) that overcome these challenges by inspecting aggregate information.

3.1 FADD: Fast Anomaly Detection given Duplicates

FADD considers identical coordinates in n-dimensional space as a super node with their duplicate count information, c_i. More specifically, rather than visiting all of the N points separately, FADD only deals with M unique super nodes $SN(u_i)$ which is the set of x_j that have the same coordinates as $u_i \in U$ (where U is the set of all unique points in n-dimensional space).

With this super node scheme, for the *kNN* algorithm, we define the density of a duplicate point $x_j \in SN(u_i)$ with more than k duplicates ($c_i = |SN(u_i)| > k$) as $lrd_k(x_j) = \frac{|SN(u_i)|}{\epsilon}$, where ϵ is a constant described in the next paragraph. This new definition of density follows naturally from Equation (1) which defines density approximately as average distance to neighbors.

In the case of points with many duplicates, the sum of the distances to other points in the neighborhood is 0. In FADD, instead of 0, we designate infinitesimal artificial distance ϵ as the sum of the distances in the super node and the average distance as the sum ϵ divided by the duplicate count c_i. The ϵ value was employed in order to avoid 0 value in the denominator; however, we still want the super node $SN(u_i)$ to have higher density than any other neighborhood set with the same

size. In order for this to hold, the ϵ value should be smaller than the sum of distances of any other neighborhood set. To achieve this the ϵ should satisfy the following condition: $\epsilon \leq k \times min(d(x_i, x_j))\ \forall x_i, x_j \in X, i \neq j$. The given condition is sufficient since the following inequality holds: $\epsilon \leq k \times min(d(x_i, x_j)) \leq \sum_{o \in N_k(p)} min(d(x_i, x_j)) \leq \sum_{o \in N_k(p)} d(p, o) \leq \sum_{o \in N_k(p)} reach\text{-}dist_k(p, o)$ for any point $p \in X$, where the rightmost term
$\sum_{o \in N_k(p)} reach\text{-}dist_k(p, o)$ is the sum of distances in the denominator of $lrd_k(p)$ in Equation (1). The c_i information differentiates the density of one duplicate point to another. For example, if we set $\epsilon = 10^{-6}$, a point with duplicate count of a thousand will have the density of 1 billion, while another point with duplicate count of 20 will have the density 20 million. Having re-defined the density, we can compute the $lof_k(x_j)$ for the duplicate point x_j, as well as the points that have the duplicate point in their k-distance neighborhood. Not surprisingly, $lof_k(u_i) = 1$ for a point with more than k duplicates, since the neighborhoods are just identical points with identical local densities. The point that encompasses the duplicate point in the k-distance neighborhood will have very high lof score, unless it has similar amount of duplicates itself. The detailed algorithm for FADD is given in Algorithm 1 .

This super node scheme gives us two benefits: (a) as shown above, we can define the duplicate point's local density and outlier score, and (b) by reducing the number of points from N to M, and by skipping the computations for duplicate points in $SN(u_i)$ that have c_i larger than k, the runtime complexity is enhanced significantly. That is, in FADD the *kNN* algorithm is dependent on the unique number of coordinates M rather than the whole space N. In terms of reduction in computation, from the statistics provided in previous section, 60% of computations are reduced by avoiding the top 5 ${c_i}^2$ computation steps. Theoretically, in heavy duplicate data sets, the runtime complexity drops from $O(max({c_i}^2))$ to $O(M)$, where M is the number of unique coordinates.

3.2 G-FADD: Grid based FADD

FADD lets us effectively compute outlier scores and defines local density of duplicate points. When using FADD, one effect caused from very high density of duplicate points is that the outlier scores for points near the heavy duplicate point become dominantly large, and wipe out all other points from marked as anomalies. Although it is true that a point near thousands and millions of duplicate points is anomalous, it does not mean that there are no other outliers with large enough $lof_k(x_i)$ values that are worth inspecting.

In the same spirit of FADD, we adopt grid based FADD, the G-FADD, in order to analyze aggregate behavior and reveal other anomalous points in multiple granularities. The granularity can be changed with the parameter l, which denotes the number of grids each dimension holds: for n-dimensional space, the number of boxes would be l^n. The basic idea is to observe the behavior of the grid rather than each point: we count the number of points that reside in the grid and only run FADD for the grids that have low count. The threshold for the count would be $k + 1$ for *kNN* algorithm, since it means that the grid is self-sufficient in querying nearest neighbors and is dense enough to be exempt from outlier detection at granularity with the number of grid l. The detailed steps of the algorithm is presented in Algorithm 2.

Algorithm 1: Fast Anomaly Detection algorithm given Duplicates (FADD)

Input: n-dimensional data points $x_1, ..., x_N$,
number of nearest neighbors k, and
constant ϵ.

Output: Outlier scores lof_1, ..., lof_N for input data points.

1: # Form a super node set $SN(u_i)$ whose elements x_j have the same coordinate with u_i,
the i^{th} unique element of X. Then, $|SN(u_i)| = c_i$ is the count of duplicates.
2: # For every super node, compute the local density $lof_k(x_j)$ for $x_j \in SN(u_i)$.
3: **for** $i = 1, ..., M$ **do**
4: **if** $|SN(u_i)| > k$ **then**
5: **for** $x_j \in SN(u_i)$ **do**
6: Assign local density: $lrd_k(x_j) \leftarrow \frac{c_i}{\epsilon}$
7: Assign outlier score: $lof_k(x_j) \leftarrow 1$
8: **end for**
9: **else**
10: **for** $x_j \in SN(u_i)$ **do**
11: Compute $lrd_k(x_j)$
12: **end for**
13: **for** $x_j \in SN(u_i)$ **do**
14: Compute $lof_k(x_j)$ using $lrd_k(x_j)$ which is obtained from line 11.
15: **end for**
16: **end if**
17: **end for**

4 Experiments

In this section, we present experimental results to answer the following questions.

- **Q1 Running Time.** How much do our algorithms reduce the running time?
- **Q2 Scalability.** How do our algorithms scale-up with data sizes?
- **Q3 Parameter.** How does the grid granularity parameter affect the running time and the found outliers in G-FADD?
- **Q4 Discovery.** What are the interesting outliers on large, real world data?

Both Q1 and Q2 are answered in Section 4.1. Q3 and Q4 are answered in Sections 4.2 and 4.3, respectively. We use the data shown in Table 3. Each data contains multi-dimensional points whose dimensions are specified in the 'Description' column. The degree, PageRank, and triangle of Twitter, US Patent, and Stack Overflow data are extracted from the Pegasus [11] package as properties of nodes in the corresponding graphs. The Weibo data contains the number of tweets, the number of followees, the location, the number of retweets, the number of comments of users in a social network. As described in Section 3.1, our algorithms are not sensitive to the choice of parameter ϵ if it is small enough. In the experiments we use $\epsilon = 10^{-6}$; changing it to the other values (10^{-5} or 10^{-7}) lead to the same results. We run the experiments on a machine with

Algorithm 2: Grid Fast Anomaly Detection algorithm given Duplicates (G-FADD)

Input: n-dimensional data point $x_1, ..., x_N$,
number of nearest neighbors k,
constant ϵ, and
number of grid l.

Output: Outlier scores lof_1, ..., lof_N for input data points.

```
# Form a super node set SN(u_i) whose elements x_j have the same coordinate with u_i,
# the i^th unique element of X. Then, |SN(u_i)| = c_i is the count of duplicates.
# Calculate local density lrd_k(x_j) for x_j ∈ SN(u_i), where |SN(u_i)| > k.
# For every super node bigger than k, compute local density lof_k(x_j) for x_j ∈ SN(u_i).
for i = 1, ..., M do
    if |SN(u_i)| > k then
        for x_j ∈ SN(u_i) do
            Assign local density: lrd_k(x_j) ← c_i / ε
        end for
    end if
end for
# Separate n-dimensional space uniformly into l^n n-dimensional bins (bin_1, ..., bin_{l^n}).
bincount_i ← number of points in bin_i
for i = 1, ..., l^n do
    if bincount_i > k then
        Assign outlier score 1 to points inside the bin_i: lof_k(x_j) ← 1 for x_j ∈ bin_i
    else
        Compute lof_k(x_j) using FADD
    end if
end for
```

2 dual-core Intel Xeon 3.00 GHz, 16 GB memory and 480 GB hard disk, running Red Hat Linux 4.1.2.

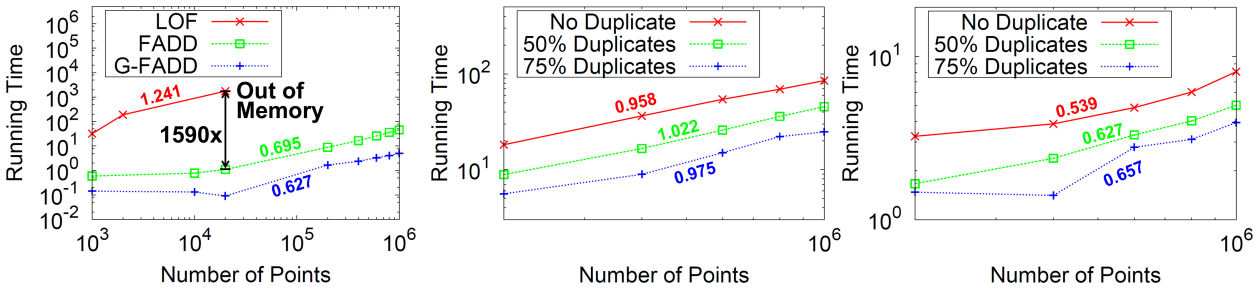
4.1 Running Time

We first show how our proposed algorithms, FADD and G-FADD, outperform the existing algorithm. Figure 3 (a) shows the running times of FADD and G-FADD, compared to LOF, on a synthetic 2-dimensional data with 50% duplicate points. Note that LOF died out of memory when the number of data points exceeds 20K. For 20K data points, FADD and G-FADD runs 1590× and 19132× faster than LOF, respectively. Also note that the running time of LOF increases quicker (slope 1.241) than the runtime of FADD (slope 0.6949) and G-FADD (slope 0.6271).

Next, we show how the ratio of duplicate points affects the running time. Figure 3 (b) and (c) show the running times of FADD and G-FADD respectively, for different ratio of duplicate points on synthetic data where the number of points ranges from 20K to 100K. Note that given the same data size, more duplicate points lead to faster running time as the size M of unique points decreases. This trend solidifies our assertion that the running time depends on M. Also, G-FADD

Data	# Dimensions	# Points	Description
Twitter 2009	3	39,972,230	degree - PageRank - triangle
US Patent	2	2,076,613	degree - PageRank
Weibo	5	2,158,558	tweets - followees - at - retweets - comments
Stack Overflow	2	243,776	degree - PageRank

Table 3: Summary of the data used. Each data contain multi-dimensional points whose dimension information is specified in the last column.



(a) FADD and G-FADD vs. LOF (b) FADD (c) G-FADD

Figure 3: (a): Running time comparison of FADD and G-FADD vs. LOF, in log-log scale. LOF died out of memory for more than 20K data points. FADD and G-FADD runs 1590× and 19132× faster than LOF, respectively. (b,c): Running time vs. data size of FADD and G-FADD in log-log scale. Note that more duplicate points lead to faster running time. Also note that G-FADD runs faster (smaller slopes) than FADD for all the ratios of duplicate points.

runs faster (smaller slopes) than FADD for all the ratios of duplicate points.

4.2 Impact of Granularity

Figure 4 shows the top 3 outliers from FADD and G-FADD with different grid granularities. Note that for all the data, the top outliers from FADD are local outliers buried in the middle of the main clouds of points because of the duplicate points in the main cloud. However, G-FADD with the coarsest grid granularity (l=8) gives global outliers which are separated from the main clouds of points. Also note that as the grid granularity becomes finer, the output from G-FADD gets closer to that from FADD.

Figure 5 shows the running time of G-FADD with different grid granularities. Note that the running time decreases dramatically as the cell width increases.

4.3 G-FADD At Work

We present interesting observations of top outliers generated from G-FADD on real world data.

Twitter degree-PageRank. Figure 4 (d) shows the top 3 outliers from G-FADD on the Twitter degree-PageRank plot. The top 3 outliers are unpopular accounts with very small degrees (7,

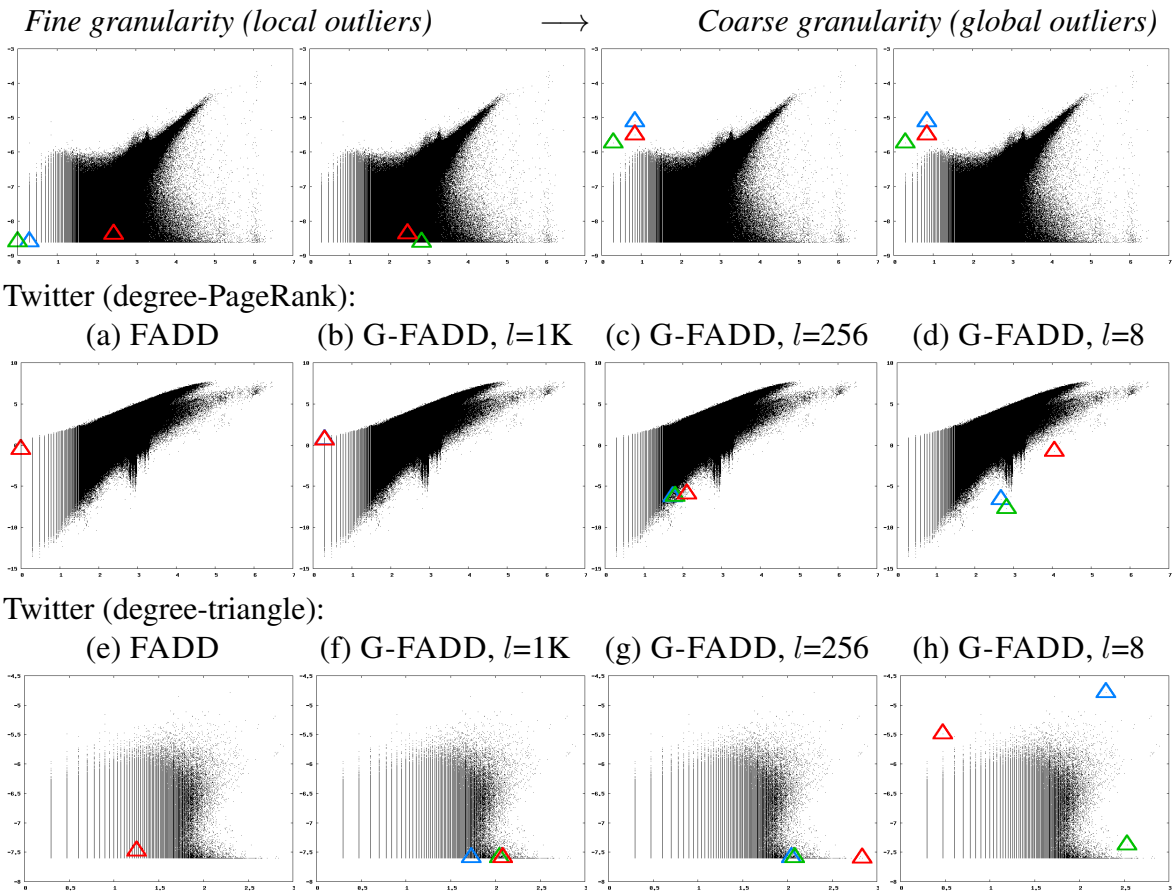
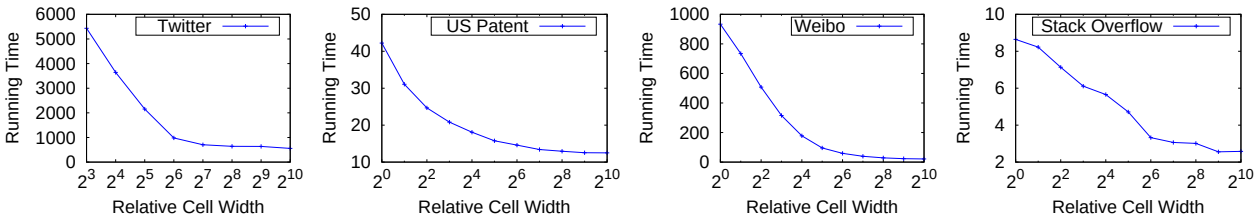


Figure 4: **[Best viewed in color.]** 2-D scatter plot highlighting top 3 outliers from FADD and G-FADD with different grid granularity l. The blue, green, and red triangles denote the points with the 1st, 2nd, and 3rd largest outlier scores, respectively. The top outliers from FADD are local outliers buried in the center of the main clouds of points due to the existing duplicate points, while G-FADD with the coarsest grid granularity (l=8) gives global outliers which are separated from the main clouds of points. As the grid granularity becomes finer, the output from G-FADD gets closer to that from FADD.



(a) Twitter (b) US Patent (c) Weibo (d) Stack Overflow

Figure 5: Running time (in seconds) of G-FADD dramatically decreases as the cell width increases.

2, and 7, respectively), but they have relatively high PageRank values which make themselves outstanding. It turns out their neighbors have relatively high degrees: the average degrees of neighbors are 1646, 89, and 343014, respectively. Due to the many neighbors, they have higher PageRanks despite their low degrees.

Twitter degree-triangle. Figure 4 (h) shows the top 3 outliers from G-FADD on the Twitter degree-triangle plot. Each point in the plot represents the degree and the number of participating triangles of a node in the Twitter who-follows-whom graph. All the top 3 outliers have relatively small number of triangles compared to their neighbors. The top outlier (blue triangle) is likely to be an advertisement spammer since it has only 3 tweets which are all about free gift card offers from Wal-Mart and Best Buy, and it has no followees at all. It has few triangles since the followers are not likely to know each other, although they had the same interest of a stroke of luck. The third outlier (red triangle) is an account of a comics character which has 11207 followers and 6 followees. It seems to have few triangles because the fans (followers) of the character might not be close friends with each other.

US Patent degree-PageRank. Figure 4 (l) shows the top 3 outliers from G-FADD on the US Patent degree-PageRank plot. The 1st and 2nd outliers (blue and green triangles, respectively) have high degrees; however the 1st outlier has much larger PageRank than the 2nd outlier. The reason is that the 1st outlier has 200 incoming edges, with no outgoing edges, and thus can absorb many PageRanks from its neighbors; however, the 2nd outlier has only 6 incoming edges and 337 outgoing edges, and thus absorbs very few PageRanks from its neighbors. The 3rd outlier (red triangle) has relatively high PageRank since all of its 3 edges are incoming edges.

5 Related Work

Two comprehensive surveys of existing outlier detection techniques can be found in [7] and [8]. The anomaly detection algorithms can be divided into categories based on the way that the data is modeled: (a) probabilistic, (b) depth-based, (c) distance-based, (d) angle-based, and (e) density-based.

The **probabilistic** approaches ([3], [21]) mark as anomalous the points that do not fit the found distribution model, while **depth-based** approaches ([22],[10]) use computational geometry to spot

outliers. Knorr and Ng introduced the **distance-based** approaches ([12], [13]), which (a) rely on the different neighborhood density between normal and outlier points, and (b) are often based on the k-nearest neighbor (kNN) distance model ([20], [4], [18], and [25]). For high-dimensional data, Kriegel et al. [15] introduced an **angle-based** approach, which tends to flag as outliers points that are at the borders, and others proposed subspace outlier detection methods ([1], [14], [17]).

The **density-based** approaches have attracted the most interest, and the representative algorithm is the Local Outlier Factor (LOF) [5]. It is based on the kNN method, and the idea is that a point is outlier if its local density is *different* from the density of its neighbors. Improvements on LOF include: INFLO [9] which handles cases of clusters that are not well separated; LOCI [19] which automatically detects outliers and micro-clusters without requiring input (k) from the user; and COF [23] that distinguishes the cases of isolation and low density.

In this work, we focus on the subtle problem of duplicate points, which increases dramatically the runtime of the distance and density-based techniques that use the kNN model. To the best of our knowledge, non of the previous works has taken care of this issue yet. We mainly study the widely used LOF method and propose duplicate-specific optimizations that render it more efficient. We do not consider the probabilistic and depth-based approaches here, since the former are bound by the choice of distribution, and the latter suffer from the curse of dimensionality. As far as angle-based methods are concerned, angle is not well defined in the case of duplicate points.

6 Conclusions

In this paper we propose FADD and G-FADD, fast and scalable algorithms that detect outliers from multi-dimensional points despite large duplicates. The main contributions are the following:

1. **No Degeneracy.** We re-design the standard outlier detection algorithm to remove degeneracy that comes from duplicate points in large, real world data.
2. **Running Time.** Our proposed algorithms enjoy near-linear running time compared to the near-quadratic running time of the existing algorithm.
3. **Discovery.** We analyze large, real world data, and find interesting outliers including Twitter accounts for advertisement spams, and nodes with high PageRanks despite small degrees.

Future research directions include the on-line outlier detection algorithms to handle duplicate points for streaming data.

References

[1] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *SIGMOD*, pages 37–46, 2001.

[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. OddBall: Spotting Anomalies in Weighted Graphs. In *PAKDD*, 2010.

[3] V. Barnett and T. Lewis. *Outliers in statistical data.* John Wiley & Sons Ltd., 2nd edition edition, 1978.

[4] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.

[5] Markus Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jrg Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[6] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Joerg Sander. LOF: Identifying density-based local outliers. In *SIGMOD Conference*, pages 93–104, Dallas, TX, 2000.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE TKDE*, 24:823–839, 2012.

[9] Wen Jin, Anthony K. H. Tung, Jiawei Han, and Wei Wang. Ranking outliers using symmetric neighborhood relationship. In *PAKDD*. Springer, 2006.

[10] Theodore Johnson, Ivy Kwok, and Raymond T. Ng. Fast computation of 2-dimensional depth contours. In *KDD*, pages 224–228, 1998.

[11] U Kang, Charalampos Tsourakakis, and Christos Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. In *ICDM*, 2009.

[12] Edwin M. Knorr and Raymond T. Ng. A unified approach for mining outliers. In *CASCON*, page 11, 1997.

[13] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[14] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In *PAKDD*, pages 831–838, 2009.

[15] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *KDD*, pages 444–452, 2008.

[16] Chao Liu, Fan Guo, and Christos Faloutsos. Bbm: bayesian browsing model from petabyte-scale data. In *KDD*, 2009.

[17] Emmanuel Müller, Matthias Schiffer, and Thomas Seidl. Adaptive outlierness for subspace outlier ranking. In *CIKM*, pages 1629–1632. ACM, 2010.

[18] Gustavo Henrique Orair, Carlos Teixeira, Ye Wang, Wagner Meira Jr., and Srinivasan Parthasarathy. Distance-based outlier detection: Consolidation and renewed bearing. *PVLDB*, 2010.

[19] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, pages 315–326, 2003.

[20] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD*, 29(2):427–438, May 2000.

[21] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., 1987.

[22] Ida Ruts and Peter J. Rousseeuw. Computing depth contours of bivariate point clouds. *Computational Statistics & Data Analysis*, 23(1):153–168, November 1996.

[23] Jian Tang, Zhixiang Chen, Ada W. Fu, and David W. Cheung. Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In *PAKDD*, pages 535–548, 2002.

[24] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *SIGMOD*, 2010.

[25] Ke Zhang, Marcus Hutter, and Huidong Jin. A new local distance-based outlier detection approach for scattered real-world data. PAKDD, pages 813–822, 2009.