Distributed Top-N Local Outlier Detection in Big Data

Yizhou Yan
Computer Science
Worcester Polytechnic Institute
Worcester, MA, USA
yyan2@cs.wpi.edu

Lei Cao
CSAIL
Massachusetts Institute of Technology
Cambridge, MA, USA
lcao@csail.mit.edu

Elke A. Rundensteiner
Computer Science
Worcester Polytechnic Institute
Worcester, MA, USA
rundenst@cs.wpi.edu

***Abstract*—The concept of Top-N local outlier that focuses on the detection of the N points with the largest Local Outlier Factor (LOF) score has been shown to be very effective for identifying outliers in big datasets. However, detecting Top-N local outliers is computationally expensive, since the computation of LOF scores for all data points requires a huge number of high complexity k-nearest neighbor (kNN) searches. In this work, we thus present the first distributed solution to tackle this problem of Top-N local outlier detection (*DTOLF*). First, DTOLF features an innovative safe elimination strategy that efficiently identifies dually-safe points, namely those that are guaranteed to (1) not be classified as Top-N outliers and (2) not be needed as neighbors of points residing on other machines. Therefore, it effectively minimizes both the processing and communication costs of the Top-N outlier detection process. Further, based on the well-accepted observation that strong correlations among attributes are prevalent in real world datasets, we propose correlation-aware optimization strategies that ensure the effectiveness of grid-based partitioning and of the safe elimination strategy in multi-dimensional datasets. Our extensive experimental evaluation on OpenStreetMap, SDSS, and TIGER datasets demonstrates the effectiveness of DTOLF – up to 10 times faster than the alternative methods and scaling to terabyte level datasets.**

***Keywords*-Local Outlier Factor; Top-N; Safe Elimination;**

I. Introduction

Motivation. Outlier detection is an important data mining technique [2] that discovers abnormal phenomena, namely values that deviate significantly from the common occurrence of values in the data [10]. Outlier detection is critical for applications from credit fraud prevention, network intrusion detection, stock investment planning, to disastrous weather forecasting.

Local Outlier Factor (*LOF*) [5] is one of the most popular outlier detection methods [2]. LOF generates an outlierness score (LOF score) for each point in the dataset. The LOF score is measured based on the *relative density* of each point in relation to its local neighbors to detect outliers. Since the *relative density* reflects the local data distribution, LOF is shown to be very effective [2], [6] at handling real world large datasets which tend to be heavily skewed [16]. A variation of LOF called Top-N LOF was proposed in [12]. It leverages the insight that outliers typically correspond to only an extremely small fraction of the overall input dataset. Therefore Top-N LOF only returns *N* points with the *largest* LOF scores that represent the most extreme outliers and thus are of great importance to the application.

State-of-the-Art. Intuitively Top-N LOF can be supported by first computing the LOF scores for all points and then selecting the Top-N among them. However, computing the LOF score for all points is a high complexity process, because it requires the expensive computation of the k nearest neighbors (kNN) for each single point. The time complexity is quadratic in the number of points. As will be shown in our experiments (Sec. V), even if leveraging the state-of-the-art *distributed* pipeline framework for LOF computation (DLOF) [18], it still takes days to process a terabyte dataset. Obviously this naive two-step solution is grossly inadequate to handle big datasets.

To reduce the computation costs, a customized strategy for Top-N LOF detection was recently proposed [19]. It features an effective pruning strategy to identify the points that do not have a chance to be among the Top-N LOF outliers without computing their LOF scores. It thus saves expensive kNN search operations. However, this centralized approach obviously cannot handle a dataset that is too large to be accommodated by one single machine. Thus, the development of a highly distributed solution customized for Top-N LOF remains open.

Challenges. Designing an efficient distributed Top-N LOF approach is challenging. Intuitively an efficient distributed Top-N LOF approach can be devised if we could make the pruning strategy [19] applicable in the DLOF framework [18]. However, given a point p, even if p is declared as not an outlier without computing its kNN, p cannot be simply eliminated from the computation process. This is so because based on the LOF definition, the kNN of p might be needed by another point q when computing q's LOF score. In a centralized setting, solving this problem is straightforward. Since all points reside on the same machine, the kNN of p can be computed on demand when processing the point q. However, in a distributed setting, the kNNs of p and q might be computed on different machines. One extra step has to be introduced in the DLOF pipeline to compute p's kNN once p is needed. This will lead to prohibitive communication and computation costs in a distributed setting – much larger than the computation costs saved by avoiding the LOF score computation of p. Furthermore, the pruning strategy [19] and the DLOF pipeline framework [19] both rely on the grid-based partitioning strategy. However, grid-based partitioning is considered to be only effective in datasets with very low dimensions, such as two or three dimensions [11]. Therefore, designing a distributed Top-N LOF approach that is effective for multi-dimensional data is challenging.

Proposed DTOLF Approach. In this work, we propose the first distributed Top-N LOF approach, called *DTOLF*, that efficiently detects the Top-N local outliers in big data.

First, DTOLF features a *safe elimination* strategy that by only looking at the local distribution characteristics of the data partition residing on one machine, efficiently identifies the *safe-to-eliminate* points that are guaranteed to not be outliers and also not be needed by any other machine. These points can be completely eliminated from the Top-N LOF computation process, thus avoiding costs due to computation, storage and transmission to other machines in the next step of Top-N LOF computation. This saves significant computation and communication costs. The key innovation here is that, given a partition $\mathbb{P}$, the safe-to-eliminate points in $\mathbb{P}$ can be correctly identified based on their distances to the boundaries of $\mathbb{P}$. This inspires us to design a safe elimination strategy that can locate the safe-to-eliminate points of each partition without requiring any information about other partitions.

Moreover, DTOLF features a correlation-aware strategy that ensures the effectiveness of DTOLF in multi-dimensional data. Utilizing the strong correlations often present in real world datasets, it effectively divides multi-dimensional data into grid partitions with a small domain range on each dimension yet avoiding the generation of a large number of partitions – critical for reducing the communication costs of distributed LOF computation and the effectiveness of the safe elimination strategy. The key insight here is that the correlations can be explored to bound the neighborhood of each data partition by only taking the essential data attributes into consideration.

DTOLF is inherently parallel and works in virtually any distributed computing infrastructure. This helps to assure ease of adoption by others on popular open-source distributed infrastructures such as MapReduce [1] and Spark [20]. Leveraging a moderate size compute cluster, DTOLF is shown to scale to big datasets at the terabyte level and cut down the processing time from days to hours.

Contributions. Key contributions of this work include:

• We propose *the first* distributed Top-N LOF approach scalable to terabyte level datasets.

• Our *safe elimination* strategy effectively identifies points from local data partition that are guaranteed to be not Top-N outliers, plus also not needed by other machines. Significant computation and communication costs are saved.

• DTOLF efficiently processes multi-dimensional data by exploiting the correlations among the data for optimizing data partitioning and the safe-elimination process.

• Experiments on real OpenStreetMap, SDSS and TIGER datasets demonstrate that DTOLF drives down the processing time of terabyte level datasets from days to hours – finally making Top-N local outlier detection practical in big data applications.

II. Preliminaries

A. Top-N LOF Semantics

Local Outlier Factor (LOF) [5] introduces the notion of *local outliers* important for many applications. More precisely, for each point p, LOF computes the ratio between its local density and the local density around its neighboring points. This ratio assigned to p as its local outlier factor (LOF score) denotes its degree of outlierness. LOF depends on a parameter *k*. For each point *p* in dataset *D*, *k* is used to determine *k-distance* and neighborhood of *p*. The *k* points closest to *p* are the *k-nearest neighbors* (kNN) of *p*, also called *k-neighborhood* of *p*. *k-distance* of *p* is the distance to its kth nearest neighbor. The LOF score below depends on the points in its k-neighborhood.

Definition 2.1: The **reachability distance** of point p w.r.t. point q is defined as:

$$\textit{reach-dist(p,q)} = \textit{max}\ \{\textit{k-distance(q), dist(p,q)}\}$$

If one of the kNN of p, say q, is far from p, the $reach$-$dist$ between p and q is simply their actual distance. On the other hand, if q is close to p, the $reach\text{-}dist$ between them is the *k-distance* of *q*.

Definition 2.2: The **local reachability density (LRD)** of a point p is the inverse of the average reachability distance of p's kNN defined by:

$$LRD(p) = 1/[\frac{\sum_{q \in Knn(p)} reach - dist(p,q)}{||\textit{k-neighborhood}||}]$$

Essentially, the LRD of a point p is an estimation of the density at point p. It represents the inverse of the average reachability distance between p and the points in its k-neighborhood. Based on LRD, LOF is defined as follows.

Definition 2.3: The LOF score of a point p is defined by:

$$LOF(p) = \frac{\sum_{q \in kNN(p)} \frac{LRD(q)}{LRD(p)}}{||\textit{k-neighborhood}||}$$

LOF score is a positive value. Intuitively, the higher the LOF score, the more the point is considered to be an outlier.

Finally, we define the semantics of Top-N LOF detection.

Definition 2.4: Given the input parameters k and N, the outliers O of a dataset D correspond to a subset of D ($O \subset D$) with cardinality $|O| = N$, where for any $p \in O$ and any $q \in D - O$, $LOF(p) \geq LOF(q)$.

B. Distributed LOF Computation Framework

In [18], a distributed framework that computes LOF scores for all data points in a given dataset D is proposed, called DLOF. DLOF first divides the data space into regular shaped grid cell partitions. Then given a partition $\mathbb{P}$, DLOF augments $\mathbb{P}$ with the "supporting area" that could potentially contain the kNNs of any of the points (called core points) in $\mathbb{P}$. As shown in Fig. 1, partition $\mathbb{P}_1$ is augmented with the gray area in partition $\mathbb{P}_2$ as its supporting area. The points inside the supporting areas are replicates of the cores points in other partitions. For example, points in the gray area serve as core points in partition $\mathbb{P}_2$ as well as support points in partition $\mathbb{P}_1$. By this, each partition is augmented to become self-sufficient, i.e., the computation of kNN can be achieved independently in each partition without having to access the data assigned to other partitions.

Given these support-augmented partitions, DLOF computes the LOF score for each point in 3 steps.

• Step 1: K-distance Computation. This step is further decomposed into two sub-steps: core partition kNN search and support partition kNN search. The core partition kNN computes the local kNNs for points in each partition utilized to bound the supporting area. The final kNN and *k-distance* of each point are computed in the support partition kNN search step and materialized as intermediate values.

• Step 2: LRD Computation. By Def. 2.1, the *reachability distances* of each point p to its kNN q is computed using the *k-distances* of p and q from step 1. At the same time, the LRD value of p − the average reachability distance of p to its kNN q, can be naturally derived and materialized.

• Step 3: LOF Computation. LRD values materialized in the second step is leveraged to compute the final LOF scores.

One critical task accomplished at each of the first two steps is that each input data point p is enriched with its corresponding intermediate values (its kNN, k-distance, LRD) computed by that step. Then at the beginning of the next step, these enriched data points are distributed again to the machines that list them as support points. This ensures each point has sufficient information to conduct the next step computation. For example, in the LRD computation step, given a core point p in partition $\mathbb{P}_1$, suppose p has a kNN q in the supporting area of $\mathbb{P}_1$. The LRD computation of p needs the kNN of q. Since q is a core point of partition $\mathbb{P}_2$, the kNN of q might be computed in a different machine at the first step and hence is not directly accessible by p. The above mechanism solves this problem by re-distributing the enhanced q to $\mathbb{P}_1$. This ensures that DLOF process is fully distributed in each step.

Moreover, DLOF has been further enhanced with an early termination strategy, thus called *DLOF-Early*. That is, instead of calculating the LOF score strictly step by step, *DLOF-Early* aggressively pushes the LOF computation into the early stage of the pipeline and completes the LOF computation of any point as early as possible. Only the point that cannot acquire its LOF score in the current step will be passed to the next step. For the details please refer to [18].

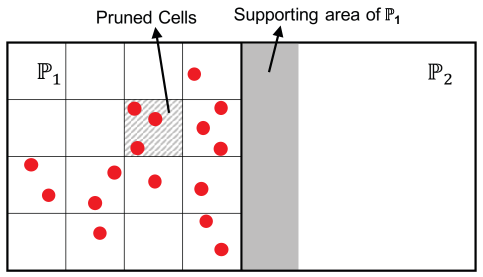


Figure 1: Support area of $\mathbb{P}_1$ and pruning (k=3).

C. TOLF: Top-N LOF approach

Recently, a centralized strategy for Top-N LOF approach is proposed, called TOLF [19]. TOLF features a *multi-granularity pruning strategy* that quickly locates and prunes points having no chance to be in the Top-N outlier list. By partitioning the data into grid cells with the length of each side set as l (Equation 1), a cell C that contains more than k points can be immediately pruned without any further evaluation. This cell-based pruning strategy is called *Cell Prune* or in short *CPrune*.

$$l = \frac{ct \times cp}{2\sqrt{d}} \tag{1}$$

In Equation 1, cp represents the distance of the closest pair of points in a data partition of a d-dimensional dataset D. ct denotes the n-th largest LOF scores among the points that have been processed so far. The intuition here is that any point in such a cell C is guaranteed to have a LOF score smaller than ct. Therefore, all points in C are not outliers. As shown in Fig. 1, partition $\mathbb{P}_1$ has been divided into small grid cells, the cell in the middle that contains 3 points can be directly pruned without any further computation.

If an entire cell cannot be pruned, then the pruning is conducted at the individual point level, thus called *Point Prune* or in short *PPrune*. The idea is to approximate the upper bound LOF score $U(p)$ (Equation 2) for a given point p. If $U(p)$ is smaller than ct, then p is not an outlier.

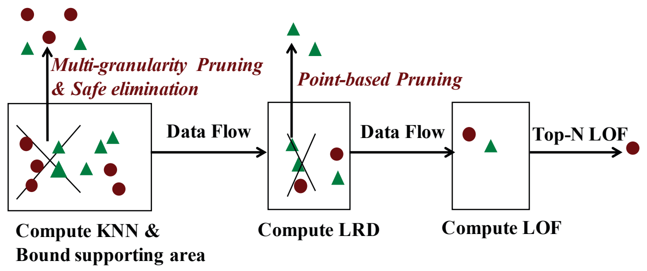
$$U(p) = \frac{max\{reach-dist(p,q)|q \in kNN(p)\}}{min\{d(q,o) \mid q \in kNN(p) \ and \ o \in kNN(q)\}} \tag{2}$$

III. Safe Elimination Strategy

In Sec. II-A, we observe that the LOF score of each single point p is determined by many other points, namely its k nearest neighbors (kNN), its kNN's kNN, and its kNN's kNN's kNN. Accordingly, each point p serves two roles during the entire computation, namely as a "core point" of which LOF value needs to be computed and a "support point" that assists the LOF computation of other core points q. Therefore, even if a point p can be pruned by TOLF [19] as a "core point" indicating that the point itself is guaranteed to be not within the Top-N LOF list, p might still need to serve as a "support point" of possibly many other points q residing on other machines thus cannot be eliminated from the detection process.

For TOLF [19], all points are available at any time on the same machine. For this reason, the computation for point p as a "support point" can be postponed and conducted only when necessary. To be specific, only when a point p is one of the kNN of point q that cannot be pruned, the kNN and *LRD* of point p will have to be calculated in order to support the computation of q's LOF score.

However, in the distributed environment, this is more complex since one machine cannot access all data points. Therefore, the kNN of a core point p in partition $\mathbb{P}_1$ cannot be computed on demand when the core point q in another machine needs it. Unfortunately introducing an additional step in the distributed framework to compute and then pass the kNN of p to the machine where q resides leads to prohibitive additional computation and communication costs. Therefore, even if a core point p in partition $\mathbb{P}_1$ can be pruned without computing its LOF score based on the key innovations from TOLF on Equations 1 and 2, we might still have to compute its kNN. However, in a distributed context, this would negate the major benefit of pruning, since the major performance bottleneck of LOF is caused by the expensive kNN search computation.



Multi-granularity Pruning & Safe elimination
Point-based Pruning
Data Flow
Data Flow
Top-N LOF
Compute KNN & Bound supporting area
Compute LRD
Compute LOF

Figure 2: New Proposed DTOLF Framework.

To solve this problem, we now propose a novel safe elimination strategy that, given a partition $\mathbb{P}_1$, can quickly identify the points guaranteed to be not needed by any other machine (so called safe-to-eliminate points) based on only the local distribution characteristics of $\mathbb{P}_1$ itself. Therefore it naturally fits the DLOF framework (Sec. II-B). Once the safe-to-eliminate points have been identified, then this opens up the opportunity to seamlessly plug the cell-based and point-based pruning strategies proposed in [19] into the DLOF framework. The overall process of our distributed Top-N LOF framework is shown in Fig. 2.

A. *Safe Elimination Strategy*

Given a partition $\mathbb{P}_i$, the points that are not the kNN of any point in any other partition $\mathbb{P}_j$ are called "safe-to-eliminate points". These points can be safely eliminated from the outlier detection process if they can also be pruned based on Equations 1 and 2.

Our safe elimination strategy is based on the observation that given a partition $\mathbb{P}_i$, a safe-to-eliminate point can be identified based on its distance to the boundaries of $\mathbb{P}_i$. Intuitively, given two partitions $\mathbb{P}_1$ and $\mathbb{P}_2$ that share a common boundary b, most likely a point p in partition $\mathbb{P}_1$ is not a kNN of any point in $\mathbb{P}_2$ if p is far away from boundary b. As shown in Fig. 3, very possibly p_2 in $\mathbb{P}_1$ is not a kNN of any point in partition $\mathbb{P}_2$.

Next, we demonstrate how our safe elimination strategy determines whether a point p in $\mathbb{P}_1$ is guaranteed not to be the kNN of one point in partition $\mathbb{P}_2$ based on its distance to the boundary. Given a point p_i in partition $\mathbb{P}_1$ and a point q_j in partition $\mathbb{P}_2$, the line that connects p_i and q_j must intersect the boundary b at point b_{ij}. For example, as shown in Fig. 3, the line connecting point p_1 in partition $\mathbb{P}_1$ and q_1 in partition $\mathbb{P}_2$ intersects the boundary at point b_{11}. The insight here is that if p_i is not a kNN of b_{ij}, then p_i is guaranteed to be not a kNN of q_j.

Lemma 3.1: Given three data points p, q, and b connected by one straight line, where $p \in$ partition $\mathbb{P}_1$, $q \in$ partition $\mathbb{P}_2$, and b is on the boundary shared by $\mathbb{P}_1$ and $\mathbb{P}_2$, if $p \notin$ kNN(b), then $p \notin$ kNN(q).

Proof: Since p is not a kNN of b, there exists k points $p_1, p_2, \ldots, p_k$ in partition $\mathbb{P}_1$, where $dist(p_i, b) < dist(p, b)$ $(1 \leq i \leq k)$. By triangle inequality, it holds that $dist(p_i, q)$ $<=$ $dist(p_i, b)$ $+dist(b, q)$ $<$ $dist(p, b)$ $+dist(b, q)$ $=$ $dist(p, q)$. Therefore, there exists k points, namely $p_1, p_2, \ldots, p_k$, that are closer to q than p. Therefore, p is not a kNN of q. Lemma 3.1 is proven. ∎

By Lemma 3.1, since p_2 in Fig. 3 is obviously not a kNN of b_{21}, it is guaranteed to also not be a kNN of q_1. Thus p_2 is called a "safe-to-eliminate point" w.r.t point q_1 in partition $\mathbb{P}_2$. Correspondingly, if a point p_i in partition $\mathbb{P}_1$ is not a kNN of any point b_{ij}, where b_{ij} corresponds to the intersection point of boundary b and the line connecting p_i and any point q_j in partition $\mathbb{P}_2$, then p_i is not a support point of partition $\mathbb{P}_2$. In other words, p_i a safe-to-eliminate point w.r.t. $\mathbb{P}_2$.

However, this raises the challenge that computing the kNNs for all b_{ij} points is expensive. Furthermore, this would require the knowledge about the points in partition $\mathbb{P}_2$ when identifying the safe-to-eliminate points in $\mathbb{P}_1$. Unfortunately, in a distributed environment, $\mathbb{P}_1$ and $\mathbb{P}_2$ might reside on different machines. Typically, in many modern shared-nothing distributed infrastructure, one machine cannot access the data on other machines as well. Even if it is possible, this will introduce prohibitive communication costs.

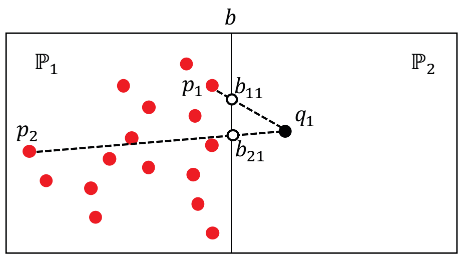


Figure 3: Boundary Points.

We solve the above problems by instead computing for each point p_i in partition $\mathbb{P}_1$ the maximal and minimal distances from p_i to any point on boundary b denoted as $p_i.max$ and $p_i.min$. First, we compute an upper bound k-distance of any point on boundary b, denote as $U(b)$, based on the k-th smallest $p_i.max$. Then given any point p_i, if its $p_i.min$ is larger than $U(b)$, it has no chance to be the kNN of any point on b. This in turn proves that p_i is a safe-to-eliminate point w.r.t. $\mathbb{P}_2$ by Lemma 3.1. Lemma 3.2 and Lemma 3.3 prove the correctness of the strategy.

Lemma 3.2: Given a partition $\mathbb{P}$ $=\{p_1, p_2, \ldots, p_n\}$ $(n \geq k)$ and a boundary b, for any point b_x on boundary b, *k-distance*$(b_x) \leq U(b)$, where $U(b)$ corresponds to the k-th smallest $p_i.max$ for $\forall p_i \in \mathbb{P}$.

Proof: Since $p_i.max = max\{dist(p_i, b_x), \forall b_x \in b\}$, $\forall b_x \in b, dist(p_i, b_x) \leq p_i.max$. Since $U(b)$ is the k-th smallest $p_i.max$, there exists at least k points $\{p_1, p_2, \ldots, p_k\}$ in partition $\mathbb{P}$ with $p_i.max \leq U(b)$ $(1 \leq i \leq k)$. Therefore, given any point b_x on boundary b, $dist(b_x, p_i) \leq U(b)$. Since $1 \leq i \leq k$, there are at least k points in $\mathbb{P}$ whose distances to b_x are smaller than $U(b)$. Therefore *k-distance*$(b_x) \leq U(b)$. Lemma 3.2 is proven. ■

Lemma 3.3: Given a partition $\mathbb{P}_1$, its boundary b shared with partition $\mathbb{P}_2$, and an upper bound $U(b)$, $\forall$ point $p_i \in \mathbb{P}_1$, p_i is a safe-to-eliminate point w.r.t $\mathbb{P}_2$, if $p_i.min > U(b)$.

Proof: Since $p_i.min = min\{dist(p_i, b_x), b_x \in b\}$, $\forall b_x \in b, dist(p_i, b_x) \geq p_i.min > U(b)$. According to Lemma. 3.2, every point on the boundary can find their kNN within distance $U(b)$. If $\forall b_x \in b, dist(p_i, b_x) > U(b)$, then p_i has no chance to be kNN of any point on boundary b. By Lemma 3.1, p_i is safe-to-eliminate point w.r.t $\mathbb{P}_2$. Lemma 3.3 is proven. ■

Since each partition corresponds to a grid cell, each boundary b is parallel to one axis of the data space. Therefore, computing $p_i.max$ and $p_i.min$ is straightforward. $p_i.max$ can be determined by first computing the distances from p_i to the endpoints of b and then selecting the largest distance. The $p_i.min$ corresponds to the perpendicular distance from p_i to b. The safe elimination algorithm based on the above observation is demonstrated in Algorithm 1.

Segment-based Safe Elimination Strategy. The effectiveness of our proposed safe elimination strategy relies

Algorithm 1 Safe Elimination

```
1: k ← number of nearest neighbors
2: $p_i.max = max\{dist(p_i, b_x), b_x \in b\}$
3: $p_i.min = min\{dist(p_i, b_x), b_x \in b\}$
4: function SAFE-ELIMINATION(POINT-LIST[$p_1, \ldots, p_m$], BOUNDARY $b$)
5:     $kNN(b)$ = Compute $p_i.max$ and maintain $k$ smallest distance
6:     $U(b) = max(kNN(b))$
7:     for $p_i \in$ point-list do
8:         if $p_i.min > U(b)$ then
9:             $S_b = S_b \cup p_i$
       return $S_b$
```

Algorithm 2 Segment-based Safe Elimination

```
1: function SAFEELIMINATIONFORSEGMENTS(POINT-LIST[$p_1, \ldots, p_m$], BOUNDARY $b$)
2:     $S_b$ = point-list
3:     $\mathbb{S} = \{s_1, s_2, \ldots, s_m\}$        ▷ divide into segments
4:     for $s_i \in \mathbb{S}$ do
5:         $S_b$ = $S_b \cup$ SAFE-ELIMINATION(point-list, $s_i$)
       return $S_b$
```

on the value of $U(b)$. When $U(b)$ is large, only a limited number of points can be marked as safe-to-eliminate points. As shown in Fig. 4, since $U(b)$ is large, the gray area corresponding to the area that potentially contains the kNNs of points on b covers almost all points in partition $\mathbb{P}_1$. Hence, only a few points can be safely eliminated.

Next, we thus design a method to minimize $U(b)$. The idea is inspired by the *boundary length* observation. Since $p_i.max$ is determined by the distances between p_i and the two endpoints of the boundary b, $U(b)$ is highly correlated to the length of boundary b. In general the longer b is, the larger the $U(b)$ tends to be. Therefore, if we divide the entire boundary into small units and compute the $p_i.max$ w.r.t each segment separately, we will get a much smaller segment level upper bound denoted as $U(s)$.

As shown in Fig. 5, the black dashed line corresponds to the $U(s)$ for segment s_3. That is, the k-th smallest $p_i.max$ from points in partition $\mathbb{P}_1$ to segment s_3. After obtaining the upper bound for s_3, the gray area is bounded, in which points are potentially the kNNs of the points on segment s_3. After establishing $U(s)$ and the corresponding kNN areas for all segments, the safe-to-eliminate points are determined, that is, the points out of the union of all gray areas. The segment-based safe elimination algorithm is shown in Algorithm 2.

In general, the safe elimination strategy not only supports the multi-granularity pruning strategy, but also reduces the duplication rate compared to DLOF [18]. More specifically, even if a point p in partition $\mathbb{P}_1$ is inside the supporting area of partition $\mathbb{P}_2$ based on the bounding method of DLOF

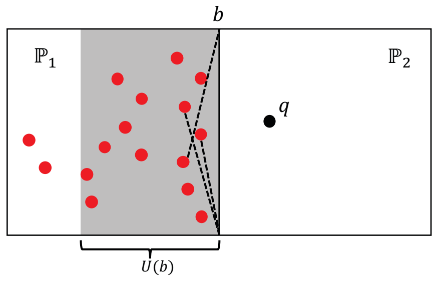


Figure 4: Large Upper Bound (k=3)

descried in Sec. II-B, it is not a supporting point of $\mathbb{P}_2$ when it is a safe-to-eliminate point in partition $\mathbb{P}_1$.

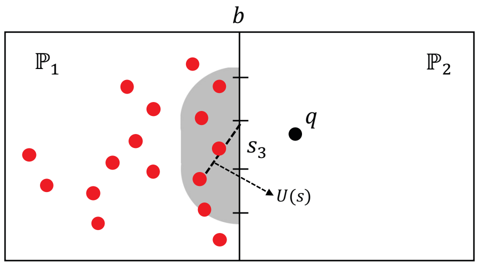


Figure 5: Segment-based Upper Bound (k=3)

Safe Elimination with Index. Note, in order to compute the $U(b)$ and identify the safe-to-eliminate points, our segment-based strategy requires the computation of p_i.max and p_i.min for each point with respect to each segment. This process can be time-consuming especially when there are a large number of points in the partition. Next, we design indexing-based method to speed up this process.

First, we show how an index such as an R-tree can be utilized to speed up the $U(b)$ computation. Assume an R-tree structure is built for partition $\mathbb{P}$. Given a segment s_i, we first locate the bounding boxes in the R-tree structure that intersect s_i. Then for each point p_i in these bounding boxes, we compute $p_i.max$ and maintain the k-th smallest $p_i.max$ as a temporal $U(b)_t$. We then construct a regular rectangle shaped "unsafe area" utilizing $U(b)_t$ as shown in Fig. 4. Then we compute $q.max$ for each point q in the bounding boxes that overlap with the unsafe area. $U(b)_t$ will be updated if $q.max$ is smaller than $U(b)_t$. $U(b)_t$ gets smaller than smaller when more points are processed. After processing all points in these bounding boxes, we get the final $U(b)$. It defines the final "unsafe area" that is smaller than the initial one.

Next, we show how to locate the safe-to-eliminate points. This is done by identifying the "unsafe points", namely the points that might be the kNNs of points in other partitions. Again we first locate the bounding boxes in R-tree that overlap the unsafe area. We then examine the data points inside these bounding boxes. Point p_i is marked as an unsafe point if $p_i.min \leq U(b)$. Naturally, the points not marked as unsafe points are marked as the safe-to-eliminate points.

Utilizing the index, only a limited number of data points is examined w.r.t each segment. Thus this index significantly speeds up the safe elimination process.

IV. Correlation-aware DTOLF

Although *DTOLF* approach works well when dealing with low dimensional datasets, as dimensionality increases, the partitioning strategy of the distributed LOF framework as well as the cell-based CPrune strategy of Top-N LOF computation become less efficient. In this section, we further propose correlation-aware optimization strategies that ensures the effectiveness of DTOLF in handling multi-dimensional data. These strategies are based on the observation that strong correlations often exist in real datasets. We now illustrate how such correlations could be leveraged during partitioning and pruning.

Correlation-aware Partitioning. Similar to other distributed analytics work [21], [15], [18], we adopt the domain-based partitioning strategy that partitions the domain space into regular shaped grid partitions. Each partition contains similar number of points. This ensures both data proximity within each partition and the load balancing across all partitions. The partitions are generated by iteratively dividing each dimension such that each dimension of a grid partition only covers a small domain range. This successfully restricts the size of the supporting area of each partition and in turn reduces the duplication rates. However this strategy tends to generate a large number of grids when the data dimension increases. Significant cost arises when assigning the points to the corresponding grids.

Correlation Observation. The observation here is that if two dimensions are strongly correlated, dividing one dimension is effectively equivalent to dividing both. For instance, given a two-dimensional domain space $x \in [0, 9]$ and $y \in [0, 9]$, if these two dimensions exhibit a strong correlation $x = y$, then when dividing dimension x into $x_1 \in [0, 4]$ and $x_2 \in [5, 10]$, dimension y tends to be also automatically divided into $y_1 \in [0, 4]$ and $y_2 \in [5, 10]$.

Based on this property, data partitioning can be performed only on independent dimensions instead of on all dimensions. This effectively reduces the number of partitions, while still ensuring that each partition has a supporting area covering a small domain space. This reduces the number of the duplicated support points and thus saves both the computation and communication cost.

Correlation-aware Pruning. When generating the cells for the cell-based *CPrune* pruning strategy, the correlation property can be leveraged in the same manner as data partitioning, that is, dividing each partition only on the independent dimensions. This will generate a smaller number of cells compared to dividing on all dimensions. Obviously when the number of the cells is reduced, on average the number of cells that contain more than k points will increase. This benefits CPrune as shown in Sec. II-C. However, by Equation 1, CPrune restricts the size of the cell on each dimension to be $l = \frac{ct*cp}{2\sqrt{d}}$. If cells are generated by dividing only on independent dimensions, the undivided dimension might not satisfy the size requirement. Therefore, the cell with more than k points cannot be simply pruned. In order to still effectively conduct CPrune on the cells produced by only dividing the independent dimensions, we enhance the original CPrune strategy with some specific constraints on correlated dimensions, namely *Correlation-aware CPrune*.

Suppose the dataset consists of a set of independent dimensions $I = \{i_1, i_2, \cdots, i_m\}$ and a set of dimensions $R = \{r_1, r_2, \cdots, r_n\}$ that are correlated with the dimensions in I. The cells are generated by only dividing independent dimensions with size *l*.

Lemma 4.1: Given a cell $\mathbb{C}_i$, all points contained in $\mathbb{C}_\beth$ can be pruned if: (1) $\mathbb{C}_i$ contains at least $k + 1$ points; (2) for each dimension $r_j \in R$, $D_{max_{r_j}} - D_{min_{r_j}} \leq l$, where $D_{min_{r_j}}$ and $D_{max_{r_j}}$ are the minimum and maximum values on dimension r_j in all points of $\mathbb{C}_i$.

Condition (2) puts the domain range constraint on the correlated dimension r_j. This ensures that the size of dimension r_j does not exceed the cell size requirement (Equation 1) of cell-based *CPrune*. Therefore, any cell satisfying the above conditions can be safely pruned despite the fact that correlated dimensions are not partitioned. Based on our correlation observation, since partitioning one dimension in effect also partitions its strong correlated dimension, the chance that Condition (2) is satisfied tends to be high. This is also confirmed in our experiments on real datasets.

V. Experimental Evaluation

A. Experimental Setup & Methodologies

Experimental Infrastructure. All distributed experiments are conducted on a Hadoop cluster with one master node and 24 slave nodes. Each node consists of 16 core AMD 3.0GHz processors, 32GB RAM, 250GB disk. Nodes are interconnected with 1Gbps Ethernet. Each server runs Hadoop 2.4.1. Each node is configured with up to 4 map and 4 reduce tasks running concurrently, sort buffer size set to 1GB, and replication factor 3. The centralized algorithm runs on a computer with Intel 2.60GHz processor, 500GB RAM, and Ubuntu operating system. All code used in the experiments is available at https://github.com/yizhouyan/TopNLOFPruningWithPrioirty.

Datasets. We evaluate our proposed methods on three real-world datasets: OpenStreetMap [9], SDSS [8] and TIGER [7] and two terabyte level synthetic datasets.

OpenStreetMap, one of the largest real datasets publicly available, contains geolocation data from all over the world and has been used in other similar research work [21], [15]. Each row in this dataset represents an object like a building or road. To evaluate the robustness of our methods for diverse data sizes, we construct hierarchical datasets of different sizes: *Partial Massachusetts* (3 million records), *Massachusetts* (30 million records), *Northeast of America* (80 million records), *North America* (0.8 billion records), up to the *whole planet* (3 billion). In our experiment two attributes are utilized for distance computation, namely *longitude* and *latitude*.

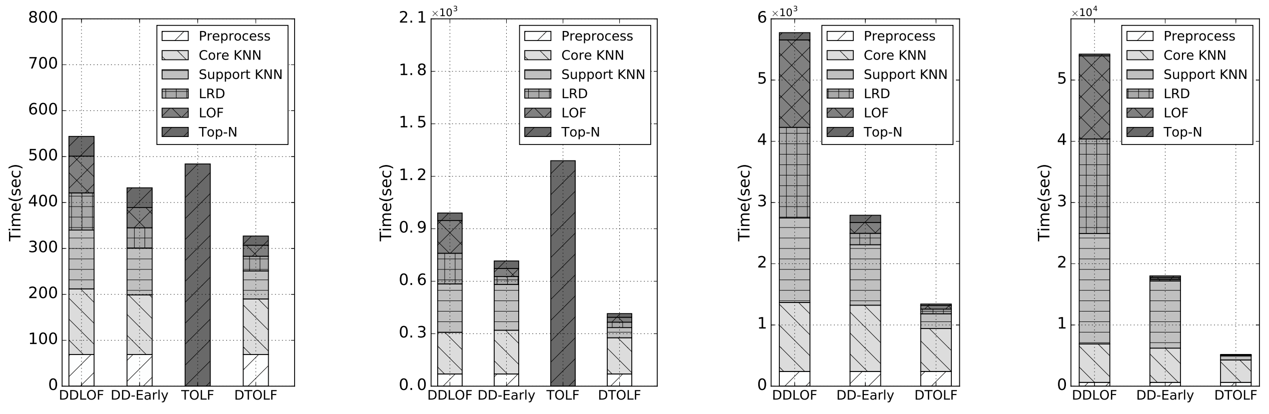
Sloan Digital Sky Survey (*SDSS*) dataset [8] is one of the largest astronomical catalogs publicly accessible. The thirteenth release of SDSS data utilized in our experiments contains more than 1 billion records and is 3.4TB in size. In this experiment we extract the eight numerical attributes including ID, Right Ascension, Declination, three Unit Vectors, Galactic longitude and Galactic latitude. The size of the extracted dataset is 240GB.

TIGER [7] dataset represents GIS features of the US. This 60GB dataset contains 70 million line segments. The four numerical attributes we utilize include the *longitude* and *latitude* of the two endpoints of each line segment.

Synthetic Datasets. The *OpenStreetMap* data for the entire planet contains more than 500GB of data. To evaluate how our proposed methods perform on terabyte level data we generate two datasets 1TB and 2TB respectively based on the *OpenStreetMap* dataset. More specifically, we generate the 2TB dataset by moving each point vertically, horizontally, and also along both directions to create three replicas. That is, given a two dimensional point $p(x, y)$ where $0 \leq x \leq d_x$ $0 \leq y \leq d_y$, three replicas $p'(x + d_x, y)$,$p''(x, y + d_y)$ and $p'''(x + d_x, y + d_y)$ are generated. Then the 1TB dataset (6 billion records) is generated by extracting half of data from the 2TB dataset (12 billion records).

Metrics. First, we measure the total ***end-to-end execution time*** elapsed between launching the program and receiving the results – common for the evaluation of distributed algorithms [21], [15], [18]. To provide more insight into potential bottlenecks, we break down the total time into time spent on key phases of the MapReduce workflow, including *preprocessing*, *core partition kNN search*, *support partition kNN search*, *LRD* calculation, *LOF* calculation and *Top-N LOF* calculation. Second, we measure the ***duplication rate*** of each method.

Algorithms. Since no prior work on distributed Top-N LOF exists in the literature, we create two alternative approaches leveraging the distributed LOF computation framework (DLOF) [18]. That is, we first compute the LOF value for each point by utilizing the DDLOF or the DD-Early approaches of [18] and then rank the points based on their LOF values. These two approaches are called DDLOF and DD-Early in our experiments. The centralized TOLF algorithm proposed in [19] is also evaluated on smaller datasets. We compare DDLOF, DD-Early, and TOLF against DTOLF proposed in our work. All four algorithms produce the exactly identical set of Top-N LOF outliers.



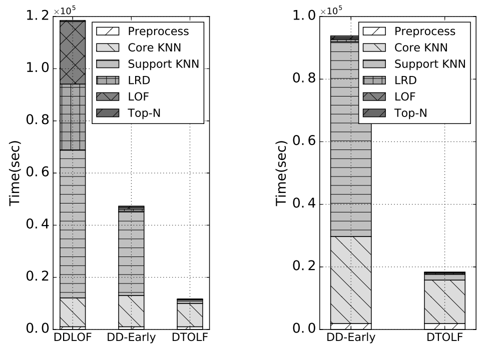
(a) Massachusetts (b) NorthEast (c) North America (d) Planet

Figure 6: Evaluation of Processing Time with OpenStreetMap Datasets.

Experimental Methodology. We conduct experiments to evaluate the **effectiveness** of our proposed algorithm using various datasets derived from the OpenStreetMap, SDSS, and TIGER datasets. Except for the experiments of varying parameter *k*, the input parameter *k* of LOF is fixed as 6 shown to be effective in capturing outliers in [5]. Similarly, except for the experiment varying parameter *n*, the input parameter *n* of top outliers is set to be 0.0001% of the total data points for each dataset. For example, in the OpenStreetMap Planet dataset (3 billion records) experiment, the top 3000 buildings with largest LOF scores are returned.

B. Evaluation of the Processing Time

We evaluate the breakdown of the processing time of the four algorithms using OpenStreetMap datasets. Results are shown in Fig. 6. *DTOLF* significantly outperforms DDLOF and DD-Early in all four cases up to 10 times and 5 times in total processing time respectively. Better yet, the larger the dataset, the more it wins. The performance gain of DTOLF results from the safe elimination strategy that eliminates the points that are neither Top-N outliers nor support points of other partitions in the k-distance computation step of the distributed pipeline. Although determining safe-to-eliminate points introduces extra costs, our indexing-based safe elimination strategy significantly speeds up the process. On the contrary, the DDLOF and DD-Early approaches cannot eliminate points, because they are not able to assess whether a point will be needed by other partitions without the support of our safe elimination strategy. This introduces extra disk IO and communication costs. Worst yet, it also increases the computation costs in the subsequent steps, since more points are involved as support points. Besides, *DTOLF* also outperforms the centralized *TOLF* approach on both Massachusetts and Northeast datasets up to 3 times in total processing time. This confirms the efficiency of the distributed strategy of DTOLF. The centralized *TOLF* approach



(a) Planet-1T (b) Planet-2T

Figure 7: Evaluation of Processing Time with Terabyte-level Datasets

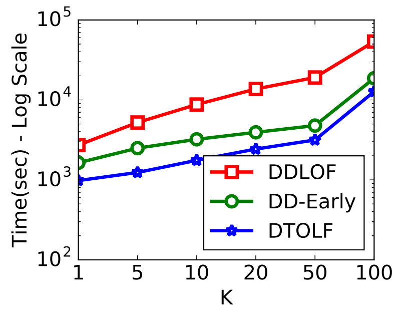
fails to process large datasets such as NorthAmerica dataset due to the memory limitation.

C. Evaluation of Scalability

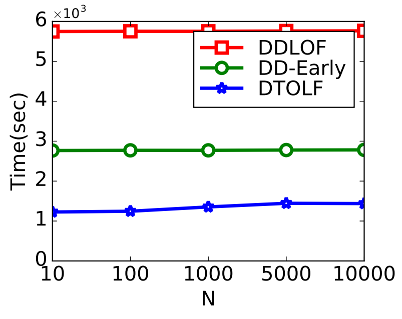
We utilize the 1TB and 2TB data described in Sec. V-A to evaluate the scalability of *DTOLF* to terabyte level data. As shown in Fig. 7, *DTOLF* is 10 times faster than *DDLOF* and 5 times faster than *DD-Early* respectively. This again is achieved by our safe elimination strategy that is able to eliminate none Top-N outlier points that are not needed by other partitions. In particular, *DTOLF* eliminates 87.88% (for 1T case) and 85.19% (for 2T case) points after the kNN search. Furthermore, as shown in Fig. 9, due to the elimination capability, the duplication rate of *DTOLF* is up to 20 times smaller as compared to the duplication rate of *DDLOF* and *DD-Early*

D. Evaluation on Various Dimensional Data

To study the impact of varying the number of dimensions on the distributed Top-N LOF outlier detection, we evalu-



(a) Vary K



(b) Vary N

Figure 8: Vary Parameters k and n on North America dataset

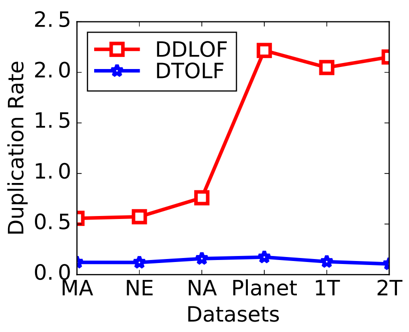
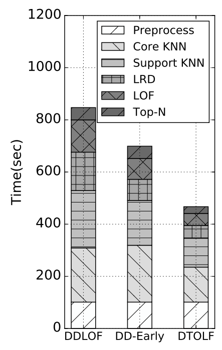
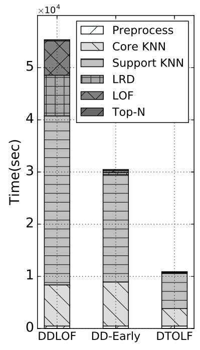


Figure 9: Duplication Rate of OpenStreetMap Datasets



(a) TIGER



(b) SDSS

Figure 10: Evaluation of Processing Time with Multi-dimensional Datasets

ate D-TOLF on the SDSS dataset (8-dimensions) and the TIGER dataset (4-dimensions).

Fig. 10(b) showcases the results on the eight dimensional SDSS dataset. *DTOLF* is around 4 times and 2 times faster than *DDLOF* and *DD-Early* respectively in total processing time. With the correlation-aware partitioning and CPrune strategies presented in Sec. IV, DTOLF is effective in multi-dimensional datasets. The performance gain of *DTOLF* can be explained by the use of smaller number of partitions and large cells that are critical to effectively identify the points guaranteed to be not Top-N outliers.

Similarly, *DTOLF* is around 2 times faster than both *DDLOF* and *DD-Early* on the TIGER dataset as shown in Fig. 10(a). In summary, our experiments on the SDSS dataset and TIGER dataset demonstrate that our DTOLF approach efficiently support datasets with varying dimensions.

E. Evaluation of Duplication Rate

Next we evaluate the duplication rates of all 3 algorithms using the same data and setting as in Sec. V-B. Since the duplication rates are identical for both DDLOF and DD-Early methods, we only show DDLOF.

Fig. 9 shows the results on the OpenStreetMap datasets. *DDLOF* has consistently higher duplication rate than *DTOLF* – up to 20 on the 2T dataset. This is expected, although DDLOF is able to acquire a relatively small supporting area for each partition by utilizing the "local" *k-distance* generated in the core partition kNN search phase (Sec. II-B), the DTOLF further reduces the number of support points by removing the safe-to-eliminate points as explained in Sec. III.

F. Evaluation of the Impact of Varying Parameters

We next evaluate the influence of the number of neighbors k and the number of outliers n. We use OpenStreetMap North America dataset (800 million records).

Varying Parameter k. Fig. 8(a) presents the results of varying the LOF input parameter k from 1 to 100. *DTOLF* outperforms alternatives up to 5 times in total processing time. As k increases, the costs of the kNN search will also increase and hence the overall processing time. However the processing time of DTOLF increases slower than *DDLOF* and *DD-Early*. Therefore the larger k is, the more DTOLF wins. This is so because as k increases, more kNN information will be written out to HDFS for each point. With the safe elimination strategy, points that are neither within the Top-n LOF list nor other partitions support points are completely removed from the process by the end of the kNN search phase - thus saving more communication and disk IO costs.

Varying Parameter n. Fig. 8(b) shows the total processing time when varying the input parameter *n*, that is, the number of outliers. N is varied from 10 to 10,000. *DTOLF* beats *DDLOF* and *DD-Early* in all cases up to 5 fold. Further, the processing time of DTOLF increases slightly as *n* increases. This indicates that the pruning and indexing of DTOLF are still very effective with large n. The processing time of *DDLOF* and *DD-Early* is stable when *n* increases. Although the sorting phase becomes more expensive as *n* increases, the costs of the sorting phase are minor compared to the LOF score computation costs.

VI. Related Work

LOF. Breunig et al. [5] proposed the notion of local outliers in contrast to global distance-based outliers [13], [17]. They defined a degree of outlierness based on the density of a point relative to its neighbors, the so called Local Outlier Factor (LOF). LOF has been shown to provide better accuracy in anomaly detection compared to global methods [14].

Top-N LOF. The concept of Top-N LOF outliers was introduced in [12]. The proposed (centralized) detection algorithm finds the Top-N LOF outliers without having to first compute the LOF score for each point. However, it requires an expensive preprocessing step – shown to be ineffective in handling large datasets. The recently proposed multi-granularity pruning strategy [19] further prune points guaranteed to not be Top-N outliers, yet it being a central strategy still cannot scale to terabyte level datasets.

Distributed LOF Computation. In [18], a distributed LOF computation approach, called DLOF, was designed. As described in Sec. II-B, DLOF computes the LOF score of each point in a fully distributed fashion. It can be leveraged to support Top-N LOF outlier by first computing the LOF scores for all points and ranking the points based on their LOF scores. However, as shown in our experiments, this takes days to process terabyte level datasets and hence cannot meet the response time requirement of modern outlier detection applications. This is caused by having to conduct the expensive LOF computation on each point.

Distributed kNN-based Outlier Detection. Distributed algorithms have been proposed for kNN-based outlier semantics. In these semantics, the outliers are the n points that have the largest k nearest neighbor distances.

In [3], Angiulli et al. utilized a solving set of points sampled from the original dataset to approximate whether a given point p is an outlier by comparing p to only the elements in this sample set. This method provides an approximate result, whereas we instead focus on providing an exact solution for the Top-N LOF outlier problem. Furthermore, [3] requires the solving set to be broadcasted to each node. This is not scalable to big datasets.

Bhaduri et al. [4] developed a distributed algorithm on a ring overlay network architecture with m machines. Their algorithm passes data blocks around the ring - allowing the computation of neighbors to proceed in parallel. Along the way, each point's neighbor information is updated and distributed across *all* nodes. For this, a central node is utilized to maintain and update the top-n points with the largest k nearest neighbor distances. Their strategies, such as checking the test blocks in a round robin fashion (requiring m iterations), is clearly not practical for shared nothing infrastructures. Furthermore, shared nothing infrastructures such as MapReduce do not feature such a central node.

VII. Conclusion

Top-N Local Outlier Factor semantics (*LOF*) is shown to be very effective in detecting outliers [12]. However existing centralized techniques cannot scale to terabyte level datasets. In this work, we propose the first distributed Top-N *LOF* outlier detection approach called DTOLF. Innovations include a safe elimination strategy that completely eliminates the points that have no chance to be the Top-N outlier set from the distributed computation process and correlation-aware partitioning and pruning that scale DTOLF to multi-dimensional datasets by leveraging the correlations among attributes of the datasets. Our experimental evaluation on OpenStreetMap, TIGER, and SDSS datasets demonstrate the scalability and efficiency of DTOLF - up to 10 times faster than the alternative approaches in handling datasets at terabyte scale.

References

[1] Apache hadoop. https://hadoop.apache.org/, 2015.

[2] C. C. Aggarwal. *Outlier Analysis: Second Edition*. Springer, 2017.

[3] F. Angiulli, S. Basta, S. Lodi, and C. Sartori. Distributed strategies for mining outliers in large data sets. *IEEE Trans. Knowl. Data Eng.*, 25(7):1520–1532, 2013.

[4] K. Bhaduri, B. L. Matthews, and C. R. Giannella. Algorithms for speeding up distance-based outlier detection. In *SIGKDD*, pages 859–867. ACM, 2011.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-based Local Outliers. In *SIGMOD*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM.

[6] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 4(30):891–927, 2016.

[7] A. Eldawy and M. F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363. IEEE, 2015.

[8] K. S. D. et.al. The sdss-iv extended baryon oscillation spectroscopic survey: Overview and early data. *The Astronomical Journal*, 151(2):44, 2016.

[9] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[10] D. M. Hawkins. *Identification of Outliers*. Springer, 1980.

[11] C. Ji, T. Dong, Y. Li, Y. Shen, K. Li, W. Qiu, W. Qu, and M. Guo. Inverted Grid-Based kNN Query Processing with MapReduce. In *ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh*, pages 25–32, Sept. 2012.

[12] W. Jin, A. K. Tung, and J. Han. Mining top-n local outliers in large databases. In *SIGKDD*, pages 293–298. ACM, 2001.

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

[14] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *SDM*, pages 25–36. SIAM, 2003.

[15] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1016–1027, 2012.

[16] G. H. Orair, C. H. C. Teixeira, Y. Wang, W. M. Jr., and S. Parthasarathy. Distance-based outlier detection: Consolidation and renewed bearing. *PVLDB*, 3(2):1469–1480, 2010.

[17] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, pages 427–438, 2000.

[18] Y. Yan, L. Cao, C. Kuhlman, and E. Rundensteiner. Distributed local outlier detection in big data. In *SIGKDD*, pages 1225–1234. ACM, 2017.

[19] Y. Yan, L. Cao, and E. Rundensteiner. Scalable top-n local outlier detection. In *SIGKDD*, pages 1235–1244. ACM, 2017.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, pages 15–28, 2012.

[21] C. Zhang, F. Li, and J. Jestes. Efficient parallel knn joins for large data in mapreduce. In *EDBT*, pages 38–49, 2012.