Scalable Top-n Local Outlier Detection

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

Local Outlier Factor (LOF) method that labels all points with their respective LOF scores to indicate their status is known to be very effective for identifying outliers in datasets with a skewed distribution. Since outliers by definition are the absolute minority in a dataset, the concept of Top-N local outlier was proposed to discover the n points with the largest LOF scores. The detection of the Top-N local outliers is prohibitively expensive, since it requires huge number of high complexity k-nearest neighbor (kNN) searches. In this work, we present thefi rst scalable Top-N local outlier detection approach called TOLF. The key innovation of *TOLF* is a multi-granularity pruning strategy that quickly prunes most points from the set of potential outlier candidates without computing their exact LOF scores or even without conducting any kNN search for them. Our customized density-aware indexing structure not only effectively supports the pruning strategy, but also accelerates the kNN search. Our extensive experimental evaluation on OpenStreetMap, SDSS, and TIGER datasets demonstrates the effectiveness of TOLF – up to 35 times faster than the state-of-the-art methods.

KEYWORDS

Local Outlier Factor; Top-N; Pruning Strategy

ACM Reference format:
Yizhou Yan, Lei Cao, and Elke A. Rundensteiner. 2017. Scalable Top-n Local Outlier Detection. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098191

1 INTRODUCTION

Motivation. Outlier detection is an important data mining technique [3] that discovers abnormal phenomena, namely values that deviate significantly from the common occurrence of values in the data [12]. Outlier detection is critical for applications from credit fraud prevention, network intrusion detection, stock investment planning, to disastrous weather forecasting.

Local Outlier Factor (*LOF*) [6] is one of the most popular outlier detection methods that addresses challenges caused by data skewness. Namely, in a skewed dataset, outliers in one portion of the data may have very different characteristics compared to those in other data regions. Therefore the outlier detection methods such as distance [14] and neighbor-based techniques [4] tend to fail, because they classify points as outliers by applying one global criteria on all data uniformly regardless of their surrounding neighborhood. LOF instead utilizes the *relative density* of each point in relation to its local neighbors to detect outliers. Since the *relative density* automatically reflects the local data distribution, *LOF* is very effective at handing skewed datasets. Since real world datasets tend to be skewed [18], LOF has been shown to be superior to other algorithms in detecting outliers for a broad range of applications [3, 16].

State-of-the-Art. The popular LOF method [6] generates an outlierness score (LOF score) for each point in the dataset. This process is rather expensive because it requires k nearest neighbors (kNN) search for each point. A variation of LOF called Top-n LOF was proposed [13] that only returns to the users the n points with *largest* LOF scores. This leverages the insight that points with highest LOF scores are the most extreme outliers and thus of great importance to the application. Second, by its very definition, applications tend to be interested in only the top worst offenders, i.e., top few points with highest outlier scores. Any analyst will never be able to examine the LOF scores of all or even a large percentage of any truly big dataset.

However as confirmed in its experiments, the Top-n LOF algorithm introduced in [13] takes thousands seconds to handle a synthetic dataset smaller than 1M. Clearly it cannot scale to large datasets. Therefore, the development of highly scalable solutions for Top-n LOF is urgent.

Proposed TOLF Approach. In this work, we propose thefirst scalable Top-n LOF approach, called *TOLF*, that efficiently detects local outliers in large datasets. TOLF features a detection method that successfully discovers the Top-n LOF outliers without having tofi rst compute the LOF score for each input point. It is based on a multi-granularity pruning strategy that quickly locates and thus prunes the points having no chance to be in the Top-n outlier list. The key insight of our strategy is that by partitioning the data into regular shaped cells with a carefully designed size, a cell at its coarse granularity that contains more than k points can be immediately pruned without any further computation. If a cell cannot be pruned in its entirety, then the pruning is conducted at the individual point level within the cell's point population based on an efficient LOF score upper bound estimation mechanism. Moreover, to fully exploit the power of the multi-granularity pruning strategy on skewed datasets, we design a data-driven mechanism that automatically

*Authors contributed equally to this work.

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098191

adapts the generation of the cells to the data distribution. As a bonus, a data density-aware index structure is constructed for free that significantly accelerates the kNN search and LOF computation process for points that could not be pruned.

Contributions. The key contributions of this work include:

• We propose *the first* Top-n LOF approach scalable to large datasets.

• Our multi-granularity pruning strategy core to *TOLF* quickly excludes most of the points from the outlier candidate set without computing their LOF scores or even running any kNN search for them.

• We design a data-driven cell generation strategy as well as the density-aware indexing mechanism that together ensure the effectiveness of the pruning strategy and of the kNN search on datasets with diverse distributions.

• Experiments on real OpenStreetMap, SDSS and TIGER datasets demonstrate that TOLF outperforms the state-of-the-art up to 35 times in processing time.

2 PRELIMINARIES: TOP-N LOF SEMANTICS

Local Outlier Factor (LOF) [6] 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:

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

If one of the kNN of p, say, q, is far from p, the *reach-dist* between them is simply their actual distance. On the other hand, if q is close to p, the *reach-dist* between them is the *k-distance* of q. The *reachability distance*, as a customized distance measure, introduces a smoothing factor for a stable estimation of the local density of p.

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:

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

Essentially, the LRD of a point p is an estimation of the density at point p by analyzing the k-distance of 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:

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

Intuitively, *LOF* scores close to 1 indicate "inlier" points. 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 are a subset $O \subset D$ with cardinality n, where for any $p \in O$ and any $q \in O$ - D, $LOF(p) \geq LOF(q)$.

3 TOLF: TOP-N LOF DETECTION APPROACH

The key ideas of TOLF are inspired by the *cutoff threshold observation* as shown below.

Cutoff Threshold Observation. To detect the Top-n outliers, it is not necessary to conduct a two step process, namely first to compute the LOF score for each point and then second to sort the points based on their LOF scores. Instead the TOP-n outliers can be directly acquired *in one step* as described below.

Since there will be at most n top outliers, during the computation process TOLF maintains an outlier candidate set $\mathbb{C}$ with n highest scored outliers seen so far. The elements in $\mathbb{C}$ are sorted based on their scores. The score of the smallest point p_n in $\mathbb{C}$ is used as a *cutoff threshold ct*. Then given a new point q, if q's score is smaller than the threshold ct, q cannot be in the Top-n list and therefore is discarded immediately. On the other hand, if q's score is larger than ct, q is inserted into $\mathbb{C}$. The nth point p_n is then replaced with the current smallest scored point in $\mathbb{C}$. ct is updated accordingly. As more points are processed, larger scored points will be found. The top-n outlier set will be finalized after all points have been processed.

Our observation here is that given a new point q, to prove it is not a Top-n outlier, we do not have to know its exact LOF score. Instead if we could efficiently estimate that the LOF score of q will be smaller than the given threshold ct, then q is guaranteed to not be an outlier. Therefore we conclude that this q could be safely pruned without computing its exact LOF score. Since most points are inliers with small LOF scores, this way most points in the dataset could be quickly pruned. This would enable the outlier detection algorithm to concentrate its precious resources on precisely conducting the LOF computation for the much smaller number of potential outlier candidates, rather than spending resources on computing and recording LOF scores for the general and much larger data population. Consequently the Top-n outlier detection process will be significantly sped up due to this pruning process.

Inspired by this cutoff threshold observation, we now propose the *multi-granularity pruning* strategy that effectively yet efficiently prunes the inlier points. Moreover, we design a data-driven mechanism that by partitioning data based on its distribution characteristics dynamically adapts the pruning strategy to skewed data.

The *multi-granularity pruning* strategy consists of two pruning stages. At the first stage, inlier points are pruned at *the group granularity* without conducting any kNN search, named *CPrune* pruning (Sec. 3.2). Points that cannot be pruned by *CPrune* will go through the next pruning stage at the *individual point granularity*, namely point-based pruning, in short *PPrune*. Both pruning methods are based on the quick estimation of the *upper bound* of a point's LOF score, namely the largest possible LOF score of the point.

3.1 LOF Score Upper Bound

We first given the theorem (Theorem 3.1) introduced in [6] that defines the LOF score upper bound for a given point.

Given a point p, let $direct_{max}(p)$ denote the maximum reachability distance between p and p's k nearest neighbors, i.e.,

$$direct_{max}(p) = max\{reach - dist(p,q) | q \in kNN(p)\} \quad (1)$$

To generalize this definition to p's kNN q, let $indirect_{min}(p)$ denote the minimum reachability distance between q and q's kNN, i.e.,

$$indirect_{min}(p) = min\{reach - dist(q,o) \mid q \in kNN(p) \text{ and } o \in kNN(q)\} \quad (2)$$

.

Theorem 3.1. ***Upper bound on LOF.*** *Given a point $p \in D$, the LOF score of p can be bounded by*

$$LOF(p) \leq U(p) = \frac{direct_{max}(p)}{indirect_{min}(p)} \quad (3)$$

Intuitively the LOF score of p is determined by the ratio of the average reachability distance of p to its neighbors q denoted as $avg\_reach(p,q)$ and the average reachability distance of q to q's neighbors o denoted as $avg\_reach(q,o)$. Since $direct_{max}(p)$ represents the largest reachability distance of p to q and $indirect_{min}(p)$ represents the smallest reachability distance of q to o, replacing $avg\_reach(p,q)$ and $avg\_reach(q,o)$ with $direct_{max}(p)$ and $indirect_{min}(p)$ respectively is guaranteed to acquire a value larger than the actual LOF score of p. Therefore Theorem 3.1 holds.

For the formal proof of Theorem 3.1 please refer to [6].

Based on our cutoff threshold observation if $U(p)$ is smaller than the cutoff threshold ct used in pruning, then p can be safely pruned. However, estimating the upper bound by computing $direct_{max}(p)$ and $indirect_{min}(p)$ is still expensive. While it does not require the computation of the exact LOF score, the reachability distances between p and its kNN q and the reachability distances between q and q's kNN have to be computed. This cost thus is effectively equivalent to the precise computation of the exact LOF score.

Next, we introduce our multi-granularity pruning strategy that leverages a more efficient upper bound estimation mechanism.

3.2 Multi-granularity Pruning Strategy

A New Upper Bound. By Theorem 3.1, $U(p)$ is based on $direct_{max}(p)$ and $indirect_{min}(p)$. If we replace $indirect_{min}(p)$ in Theorem 3.1 with a smaller value, more specifically, the distance of the closest pair of points cp in D, a new upper bound $U'(p)$ can be derived which is larger than $U(p)$.

Theorem 3.2. ***New Upper Bound.*** *Given a point $p \in D$, the LOF score of p can be bounded by:*

$$LOF(p) \leq U'(p) = \frac{direct_{max}(p)}{cp} \quad (4)$$

If $U'(p)$ is smaller than the cutoff threshold ct, then the true LOF score of p is guaranteed to be smaller than ct. Therefore p can be pruned. In other words, $U'(p)$ can be utilized to safely prune inliers. Estimating $U'(p)$ is much more efficient than computing $U(p)$, since it avoids the computation of $indirect_{min}(p)$ for each individual point.

However, to acquire $U'(p)$, $direct_{max}(p)$ still has to be computed. Next we introduce our *CPrune* pruning strategy that effectively utilizes $U'(p)$ to prune inliers, while avoiding the computation of $direct_{max}(p)$. The key insight of CPrune pruning is that by partitioning the data into regular shaped cells whose size is determined by cp and ct, all points in a dense cell (with more than k points) have their $U'(p)$ guaranteed to be larger than ct. Therefore they can be immediately pruned without any further computation.

Lemma 3.3. ***CPrune Pruning.*** *Let cp be the distance of the closest pair of points in a d-dimensional data D and ct denote the LOF cutoff threshold for pruning. Now let us assume that the domain space of D is evenly divided into hyper-rectangle cells with the size of each side as $\frac{ct*cp}{2\sqrt{d}}$. Given a cell $\mathbb{C}$, all points contained in $\mathbb{C}$ can be pruned immediately if $\mathbb{C}$ contains more than k points.*

Proof. Since $cp \leq indirect_{min}(p)$, then $LOF(p) < \frac{direct_{max}(p)}{cp}$. If $\frac{direct_{max}(p)}{cp} < ct$, then point p can be pruned. This condition is equivalent to $direct_{max}(p) < ct * cp$. $direct_{max}(p)$ represents the maximum reachability distance between p and its kNN. By the reachability definition in Def. 2.1, if p can acquire its kNN's kNN within $ct * cp$, then p can be pruned.

If there are $k + 1$ or more points in $\mathbb{C}$, these points can all find their kNNs within the diagonal length of $\mathbb{C}$ $\frac{ct*cp}{2}$. Similarly the kNNs of these points can all find their kNNs within the twice diagonal length of $\mathbb{C}$, namely $ct * cp$ as shown in Fig. 1. Therefore $\forall$ point p $\in \mathbb{C}$, $direct_{max}(p) < ct * cp$. Thus all points in $\mathbb{C}$ can be pruned immediately without further evaluation.

□

Fig. 1 shows an example of *CPrune* pruning. In this case k is set as *2*. The central cell contains more than k points. Therefore all points in it can be pruned immediately.

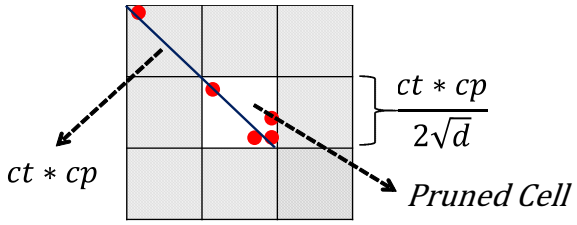


Figure 1: CPrune Pruning

Point-based pruning (PPrune). By Lemma 3.3, a cell cannot be pruned if it is not dense enough ($\leq k$ members). The points in such cells then go through the next pruning stage, namely the *point-based pruning* (*PPrune*). PPrune works in the following two steps.

First, the $direct_{max}(p)$ must be computed to acquire $U'(p)$. As shown in the proof of Lemma 3.3, if $direct_{max}(p) < ct \times cp$, $U'(p) > ct$. Therefore p is guaranteed to be not an outlier and can be pruned. Next, if $direct_{max}(p)$ does not satisfy the above condition, in order to prune more points a LOF score upper bound $U(p)_t$ tighter (smaller) than $U'(p)$ has to be computed. This can be achieved by estimating a lower bound of $indirect_{min}(p)$ tighter (larger) than cp, as defined below.

Lemma 3.4. ***Approximation of*** $indirect_{min}(p)$**.** *Given a point p in dataset D,* $indirect'_{min}(p)$ *is defined as:*

$$indirect'_{min}(p) = min\{d(q,o) \mid q \in kNN(p) \text{ and } o \in kNN(q)\} \quad (5)$$

. Then $indirect'_{min}(p) \leq indirect_{min}(p)$.

Proof. By the reach-distance definition (Def. 2.1), $reach-dist(q,o) = max(k-distance(o), d(q,o))$. Therefore $d(q,o) \leq reach-dist(q,o)$. Since $indirect_{min}(p) = min\{reach-dist(q,o) \mid q \in kNN(p) \text{ and } o \in kNN(q)\}$, $indirect'_{min}(p) \leq indirect_{min}(p)$. Lemma 3.4 holds. □

Unlike the direct computation of $indirect_{min}(p)$, $indirect'_{min}(p)$ no longer relies on computing the reachability distance reach-dist(q,o), where q is the kNN of p and o is the kNN of q. Instead it can be derived from the $d(q,o)$. Therefore computing $indirect'_{min}(p)$ avoids the computation of k-distance(o) needed by reach-dist(q,o). Since o represents p's kNN's kNN, in total there are k^2 such objects o. Therefore compared to $indirect_{min}(p)$ computing $indirect'_{min}(p)$ avoids k^2 kNN searches and thus is much more efficient .

Since $cp \leq indirect'_{min}(p) \leq indirect_{min}(p)$, replacing $indirect_{min}(p)$ with $indirect'_{min}(p)$ in Theorem 3.1 will get a bound $U(p)_t$ which is larger than $U(p)$ but smaller than $U'(p)$. Therefore $U(p)_t$ is tighter than $U'(p)$. Potentially more points will be pruned in this pruning stage by utilizing $U(p)_t$.

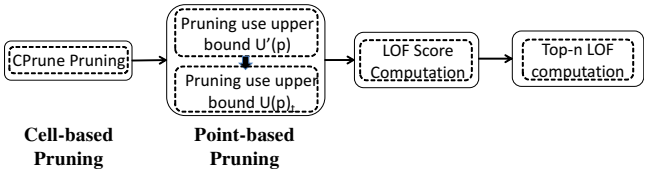


Figure 2: Overall Process of TOLF

The overall process of TOLF is shown in Fig. 2. After partitioning the dataset into cells, the CPrune pruning is first applied to prune the cells satisfying Lemma 3.3. The points in the remaining cells then go through the PPrune pruning based on Lemmas 3.2 and 3.4. The LOF score computation is only applied on the points not pruned by CPrune and PPrune, from which the Top-n outliers are derived.

4 DATA-DRIVEN TOP-N LOF DETECTION

Issues Caused by Skewed Data. However, simply applying the multi-granularity pruning strategy on a *skewed* data D and evenly dividing the *whole domain space* of D into hyper-rectangles with the size of each side as $\frac{ct*cp}{2\sqrt{d}}$ by Lemma 3.3 may cause severe issues, as shown below.

One problem concerns the generation of a large number of empty or very sparse cells when dividing the sparse area. This then would lead to significant maintenance costs without being able to reap any benefit of pruning. Even more challenging, the distance of the closest pair cp is determined by the points located in the most dense area. Therefore cp risks being very small compared to the distances of the points in other less dense areas. Since the size of the generated cells relies on cp , applying cp uniformly to the whole dataset will cause very small cells to be generated, none or very few of which will contain more than k points even in the dense area. As consequence our CPrune pruning might not be very effective.

Data-Driven TOLF: Big picture. To solve the above problem, we further enhance TOLF with a set of data-driven optimization strategies, then called data-driven TOLF, or in short *D-TOLF*. First, D-TOLF no longer applies CPrune pruning blindly over the whole dataset. Instead based on the distinct data characteristics of different areas, D-TOLF determines what areas can benefit from CPrune pruning and applies CPrune only on these areas. Furthermore, instead of generating evenly sized cells by applying the global threshold size cp of the whole dataset, cells in different areas now are assigned customized sizes adapted to the density of the data. Moreover, during the cell generation process, *D-TOLF* produces a density-aware index as side product. Unlike the traditional single tree indexing structure, it is composed of multiple trees, each of which best fits the data characteristics of one corresponding area, so called a *Forest* index. It speeds up the kNN search process in the datasets with skewed distribution as confirmed in our experiments (Sec. 5).

Overall D-TOLF is composed of two steps, namely uniform area generation (UAG) and density-aware cell generation (DCG). UAG divides the domain space of the whole dataset into *multiple areas*, each with a data distribution close to uniform. It ensures that the closest pair (cp) of each such area is not much smaller than the average distance of other point pairs. This succeeds to lead to a tighter (small) LOF score upper bound, which in turn makes the pruning effective. Next, given such a generated area, DCG decides whether this area will benefit from CPrune pruning and hence ought to be further divided into cells. If so, inspired by the QuadTree indexing, it generates cells with their size reflecting the density of their respective area. It produces a tree index customized for the sake of CPrune pruning. The trees of different areas pulled together corresponds to a *forest* index for speeding up kNN search on the whole dataset.

4.1 Uniform Area Generation

Next, we present our uniform area generation algorithm (UAG) that adopts the dual-purpose divide and conquer process to produce 'close to be uniform' areas and their closest pair distances (cp) in one step.

In **the divide phase**, similar to the typical divide and conquer based cp computation algorithm [7], *UAG* recursively divides the domain area into sub-areas, each containing at least k points. Since UAG aims to generate both uniform areas and closest pairs, unlike the closest pair algorithm, the final sub-areas as well as the intermediate areas produced during the divide process are also hierarchically maintained in a binary tree structure. Each node records the coordinates of the corresponding area's corners. The leaf node represents the final area which contains the data points. Figure 3 shows a 2-dimensional example where $k = 2$. The original area S has been divided into four sub-areas S_{11}, S_{12}, S_{21}, and S_{22}. S_1 and S_2 are the intermediate areas.

During **the conquer phase**, first, the closest pair distance is computed for each leaf sub-area. The sibling leaf sub-areas S_{i1}

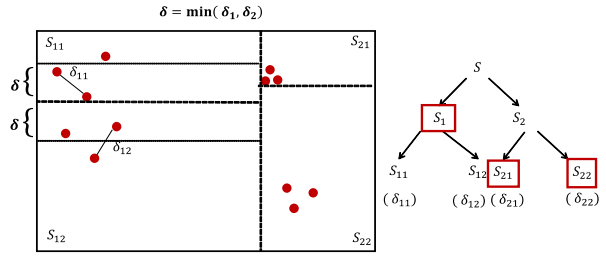


Figure 3: Divide into area – a divide and conquer process.

and S_{i2} will be merged into their parent area S_i if their closest pair distances cp_{i1} and cp_{i2} satisfy the following condition: $\frac{\max\{cp_{i1}, cp_{i2}\}}{\min\{cp_{i1}, cp_{i2}\}} < diff$, where $diff$ is a threshold used to control the difference between cp_{i1} and cp_{i2}. This ensures that the merged area S_i would not have a closest pair distance cp_i much smaller than cp_{i1} and cp_{i2}. The exact cp_i of S_i is computed similarly to the conquer process of the typical closest pair algorithm [7]. On the other hand, if S_{i1} and S_{i2} do not satisfy this merge condition, they are marked as 'final'. At the same time, their parent node S_i is marked as 'unmergeable'. This merge process recursively traverses upwards towards the root as long as a pair of sibling nodes are still mergeable and stops once no mergeable node remains. The output of UAG is the final nodes and their closest pair distances cp. It is apparent that UAG succeeds to produce the 'close to be uniform areas' as well as their cp in one step.

As depicted in Figure 3, S_{21} and S_{22} cannot be merged and are marked as final (represented as red rectangles). Their parent node S_2 is marked as unmergeable. S_{11} and S_{12} are merged into parent S_1. At the next upper layer, S_1 is marked as final, since S_1 cannot merge with S_2.

The **time complexity** of *UAG* in the worst case is as high as the classical closest pair algorithm $O(nlogn)$ with n as the number of points). This arises when the entire dataset is uniform and all nodes can be merged into one final node.

Divide Or Not. The above produced areas have different densities. A sparse area or even a dense area with a very small cp may not benefit from CPrune pruning. Hence it is not advisable to generate cells and indices for all these areas. Instead, as the first step our density-aware cell generation (DCG) algorithm determines what areas ought to be further divided into cells based on their densities and the cp computed above. By this, DCG fully unleashes the power of CPrune pruning, while avoiding unnecessary overhead.

DCG first classifies the areas into dense and sparse. If an area contains fewer than $t * k$ points where t is a tunable parameter, it is classified as sparse. And, no cell will be generated for it. On the other hand, a dense area would be divided into cells. And, an index will also be produced accordingly. The intuition here is that even if a dense area cannot benefit from *CPrune* pruning, indexing still offers significant speed up of the expensive kNN search when the number of points of this area is large.

To determine the size of the cell, DCG further classifies the dense areas into two categories based on their cp. Given an area $\mathbb{A}$ if its cp is large relative to the average distance avg_d of the point pairs in $\mathbb{A}$, e.g., $\frac{avg_d}{cp} < 3$, $\mathbb{A}$ will be divided into cells with size $S = \frac{ct*cp}{2\sqrt{d}}$ based on Lemma 3.3. In this case the cells are large and hence have a high chance to contain more than k points. This guarantees the effectiveness of CPrune pruning. The average distance avg_d of $\mathbb{A}$ is estimated utilizing the property that each area $\mathbb{A}$ produced by *UAG* is close to uniform. More specifically, suppose $\mathbb{A}$ has n two-dimensional points and covers a domain region with size $|x| * |y|$, then $avg_d \approx \sqrt[2]{\frac{|x|*|y|}{n}}$.

On the other hand, if cp of $\mathbb{A}$ is small relative to avg_d, the LOF score upper bound $U'(p), p \in \mathbb{A}$ will be large. Hence the chance of pruning points from $\mathbb{A}$ by *CPrune* is small. In this case, still setting the size of each cell as $\frac{ct*cp}{2\sqrt{d}}$ risks generating a large tree structure with many extremely small cells, each containing very few points. This inevitably leads to an increase in tree maintenance and retrieval costs. Therefore a larger cell size is preferred to ensure each cell on average contains at least k points for the sake of kNN search. Similar to the estimation of avg_d, in the two-dimensional case the size of the cell could be set as $S = \sqrt[2]{\frac{|x|*|y|*k}{n}}$.

Algorithm 1 Build FixedAreaTree

```
1:  k ← number of nearest neighbors
2:  checkNodes ← initialize Stack<Node>
3:  Root ← initialize root node
4:  function BUILDTREE()
5:      push Root into stack checkNodes
6:      while Stack checkNodes not empty do
7:          curNode ← pop one node from checkNodes
8:          if # points in curNode ≥ k + 1 then
9:              GENERATECHILDREN(curNode)
10: function GENERATECHILDREN(CURNODE NODE)
11:     childNodes − list ← divide curNode
12:     for each children c ∈ childNodes-list do
13:         if  is the same size as a small bucket then
14:             set as 'Leaf Node'
15:             if # points in  ≥ k + 1 then
16:                 set 'Can Prune'
17:             else
18:                 check if empty
19:         else
20:             if # points in  ≥ k + 1 then
21:                 push  into stack checkNodes
22:             else
23:                 set as 'Leaf Node' and check if empty
```

4.2 Density-aware Cell Generation

In summary DCG classifies all areas into three categories, namely *sparse areas* without indexing, *small cp dense areas* with indexing for kNN search only, and *large cp dense areas* with indexing for both *CPrune* pruning and kNN search.

Cell Generation and Indexing. After determining the cell size S for area $\mathbb{A}$, DCG generates cells and builds a tree index for $\mathbb{A}$ customized for CPrune pruning. Similar to the QuadTree [21], each node of the tree represents a bounding box covering some part of the space being indexed, with the root node covering the entire

area. Each leaf node corresponds to a final cell $\mathbb{C}$. However, DCG no longer requires that each internal node contains exactly 2^d children. Instead it ensures that the area covered by each leaf node corresponds to $n \times S$, so called *FixedAreaTree*. By this DCG generates cells $\mathbb{C}$ with their size guaranteed to be S if $\mathbb{C}$ contains more than k points. Such cells $\mathbb{C}$ satisfy the requirement of CPrune pruning and hence can be pruned immediately once produced as shown in Line 16 of Alg. 1.

To achieve this, DCG first evenly divides the given area into small *buckets* with the pre-determined size S. The bucket instead of the point is used as the minimal operation unit of the index construction algorithm. The construction of *FixedAreaTree* is similar to building a QuadTree [21] as a node split process. A node is defined as splittable if its buckets contain in total more than k points as shown in Alg. 1 (Line 8). Due to space restrictions, we omit the details here.

Complexity of DCG. The complexity of mapping points to buckets is $O(n)$ with n the number of points. The complexity of constructing the FixedAreaTree index is the same as the complexity of building a QuadTree, namely $O(mlogm)$ with m the number of buckets. Therefore the total complexity of DCG is $O(n + mlogm)$.

Overall Complexity of D-TOLF. Since the complexity of UAG is $O(nlogn)$, the overall preprocessing complexity of D-TOLF hence is $O(nlogn + n + mlogm)$. The number of buckets m is much smaller than the number of points n. Therefore the complexity of D-TOLF is determined by $O(nlogn)$, which is the same as the complexity of the traditional indexing. Therefore we note that little additional overhead is introduced by D-TOLF.

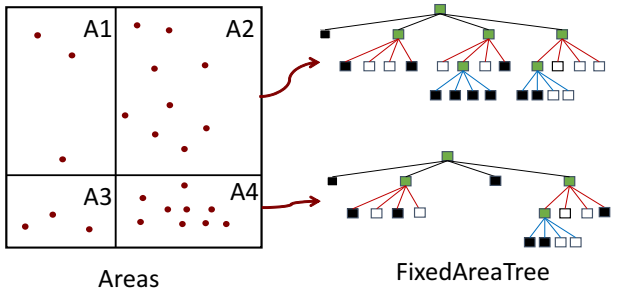


Figure 4: Forest Index

4.3 Forest Index-based kNN Search

After building the FixedAreaTree, each dense area is indexed by a tree structure as shown in Fig. 4. Therefore the whole dataset D can be represented by a *forest* index composed of multiple trees, each fitting the data characteristics of its particular area. Next, we introduce our kNN algorithm that by leveraging the forest index, speeds up the kNN search required by point-based pruning and LOF score computation, so called *ForestKnn*.

Local kNN search. Given a point p, *ForestKnn* first searches its kNN within its local area $\mathbb{A}_i$ in which p resides, called local kNN search. If $\mathbb{A}_i$ is associated with a FixedAreaTree, the traditional indexed-based kNN search mechanism could be equally applied here.

Utilizing the kNN found in $\mathbb{A}_i$, called local kNN, *ForestKnn* then determines whether the local kNN is its actual kNN within the whole dataset. More specially, if the distance of p to its kth local nearest neighbor is smaller than the shortest distance from p to any boundary of area A_i, then the local kNN is guaranteed to be the actual kNN. This is so because no point outside $\mathbb{A}_i$ can possibly be closer to p than its local kNN. If this condition does not hold, *ForestKnn* continues to search the neighboring areas of A_i using a external kNN search.

External kNN Search. The external kNN search is conducted on the trees of other adjacent areas $\mathbb{A}_j$. Unlike the local kNN search (traditional kNN search) which starts the search from the leaf node containing p, there is no leaf node in A_j containing p. Therefore the external kNN search has to first locate the leaf nodes that possibly have the kNN of p in a top-down manner from the root node, First, it checks whether a child node nd_i of the root node possibly has the kNN. If the shortest possible distance between p and the boundary of the sub-area represented by nd_i is smaller than the local k-distance of p computed within its local area A_i, nd_i could contain p's kNN. Thus the children of nd_i must be recursively evaluated in a depth first manner until all leaves underneath nd_i are traversed. The leaf nodes that possibly contain the kNN of p are marked. The kNN search then is only conducted on the points within such leaf nodes. Therefore, the CPU costs are significantly reduced.

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup & Methodologies

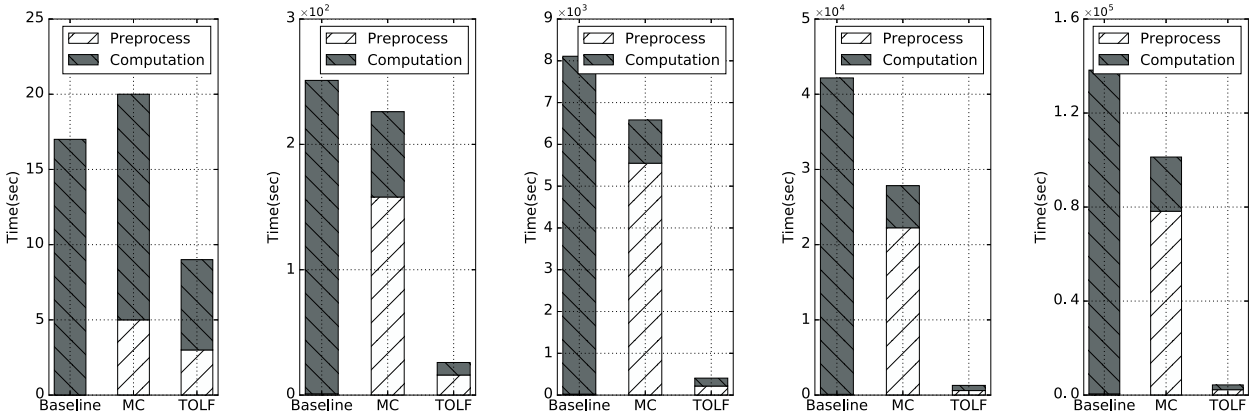
Experimental Infrastructure. All experiments are conducted on a computer with Intel 2.60GHz processor (Intel(R) Xeon(R) CPU E5-2690 v4), 500GB RAM, and 8TB DISK. It runs Ubuntu operating system (version 16.04.2 LTS). All code used in the experiments is made available at GitHub [2].

Datasets. We evaluate our proposed methods on three real-world datasets: OpenStreetMap [11], SDSS [9], and TIGER [8].

The **OpenStreetMap** dataset we use contains the geolocation of buildings all over the world. Each row in this dataset represents a building. OpenStreetMap dataset has been used in other similar research work [17, 22]. Two attributes are utilized in the experiments, namely *longitude* and *latitude* for distance computation.

To evaluate the performance of our proposed methods on various dataset sizes, we extract from OpenStreetMap five data subsets of ***different sizes***, including ***Rhode Island***, ***Connecticut***, ***Massachusetts***, ***US Northeast*** and ***US South***. The number of data points gradually grows from 0.67 million to more than 210 million.

Sloan Digital Sky Survey (*SDSS*) dataset [9] is one of the largest astronomical catalogs publicly accessible. It covers more than one third of the entire sky. We extract 100 million records from the thirteenth release of SDSS data [1] with eight numerical attributes including ID, Right Ascension, Declination, three Unit Vectors, Galactic longitude and Galactic latitude. The size of the extracted dataset is 25GB.



(a) Rhode Island (b) Connecticut (c) Massachusetts (d) US Northeast (e) US South

Figure 5: Evaluation of Processing Time with OpenStreetMap Datasets.

TIGER [8] dataset represents GIS features of the US. This 60GB dataset contains 70 million line segments. The four numerical attributes we work with include the *longitude* and *latitude* of two endpoints of the line segments.

Table 1: Summary of OpenStreetMap datasets.

	# of records	dataset size
Rhode Island	0.67 million	~30M
Connecticut	2.1 million	~ 90M
Massachusetts	31 million	~1.2G
US Northeast	81 million	~3.5G
US South	210 million	~10G

Metrics. We use the following measures. First, we measure the total ***processing time*** of each method on each dataset. To provide more insight into the preprocessing overhead, we break down the total processing time into time spent on preprocessing and Top-n outlier detection. Second, we measure the effectiveness of our proposed multi-granularity pruning strategy (Sec. 3.2) by the ratio of the number of pruned records versus the total number of records.

Algorithms. We compare the proposed methods experimentally. (1) The two step baseline method *baseline*: first compute the LOF scores of all points utilizing the algorithm in [6] and sort the points based on their LOF scores; (2) MC: the state-of-the-art Top-n LOF algorithm *MC* as described in Sec. 6; (3) *TOLF*: our proposed top-n LOF outlier detection approach. All three algorithms produce the exactly identical Top-n LOF outliers in any case.

Experimental Methodology. We conduct experiments to evaluate the **effectiveness** of our proposed algorithms using various datasets derived from the OpenStreetMap, SDSS, and TIGER datasets. Except for the experiments of varying parameter k and evaluating the pruning strategy of *MC*, the input parameter k of LOF is fixed as 6 shown to be effective in capturing outliers in [6]. 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 US South dataset experiment, the top 200 most unusual buildings are returned.

5.2 Evaluation of the Processing Time

We evaluate the breakdown of the processing time of the three algorithms using five OpenStreetMap datasets described above.

Fig. 5 demonstrates the results on the OpenStreetMap datasets. *D-TOLF* significantly outperforms the baseline solution and the state-of-the-art *MC* in all five cases up to 35 times in total processing time. Better yet, the larger the dataset, the more it wins. The performance gain of D-TOLF results from our multi-granularity pruning strategy, the data-driven cell generation mechanism, and the density-aware forest indexing. The multi-granularity pruning strategy (in Sec. 3.2) quickly prunes the points that are not possible to be top-N outliers without computing their LOF scores or even kNNs. The data-driven cell generation mechanism ensures the multi-granularity pruning works effectively on skewed datasets with various distributions. While the forest indexing created during the cell generation process significantly speeds up the kNN search for the following two reasons. First, the height of each tree in the forest index is much smaller than the height of a tree index built on the whole dataset, while it is the height of the tree that determines the costs of index-based kNN search. Second, our forest index automatically adapts to the data densities of different areas. The state-of-the-art *MC* algorithm does not clearly superior to the naive two steps baseline approach because of its heavy preprocessing costs as shown in Fig. 5. On the other hand despite the efficiency of our multi-granularity pruning strategy the preprocessing costs of D-TOLF are much lower than *MC*.

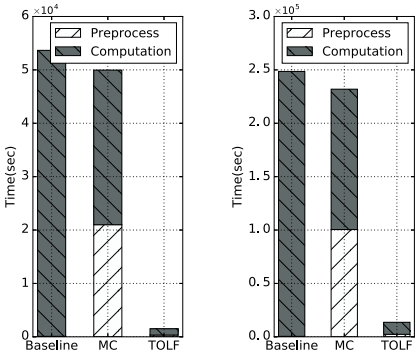
5.3 Evaluation on Various Dimensional Data

To study the impact of varying dimensions on Top-n LOF outlier detection we evaluate D-TOLF on the TIGER dataset (4-dimensions) and SDSS dataset (8-dimensions).

SDSS Dataset. Fig. 6(b) showcases the results on the eight dimensional SDSS dataset. *D-TOLF* is around 20 times faster than *baseline* and *MC* in total processing time due to taking advantage of the

Table 2: Effectiveness of Proposed Pruning Strategies (% of total number of records).

	Rhode Island	Connecticut	Massachusetts	US Northeast	US South
CPruning	11.07%	2.54%	10.39%	5.13%	11.59%
PPruning	83.07%	92.58%	81.75%	72.60%	84.41%
Total Pruning	94.14%	95.12%	92.14%	77.73%	96.00%



(a) TIGER (b) SDSS

Figure 6: Evaluation of Processing Time with Multi-dimensional Datasets

multi-granularity pruning strategy and the density-aware forest indexing (Sec. 4). Similar to the OpenStreetMap dataset experiments the preprocessing costs of MC are even larger than its Top-n LOF computation costs. Therefore MC is only slightly better than *baseline* in total processing time.

TIGER Dataset. The TIGER dataset contains longitude and latitude of two endpoints in each record. As shown in Fig. 6(a), *D-TOLF* is more than 30 times faster than *baseline* and *MC* due to the reasons similar to the SDSS dataset. Again *MC* is only slightly better than *baseline* because of its heavy preprocessing cost.

In summary, our experiments on the SDSS dataset and TIGER dataset demonstrate that our D-TOLF approach efficiently support datasets with varying dimensions.

5.4 Evaluation of the Effectiveness of Pruning

Pruning of D-TOLF. In this set of experiments we first evaluate the effectiveness of our multi-granularity pruning strategy by measuring the ratio of the number of pruned points and the total number of points in the dataset. The same data and setting are utilized as in Sec. 5.2. The results are shown in percentage.

As shown Tab. 2 (column 2-6), more than 10% of points can be pruned immediately without any evaluation by CPrune pruning. Further, up to 90% of the points can be pruned without computing their LOF scores by our D-TOLF point-based pruning (PPrune).

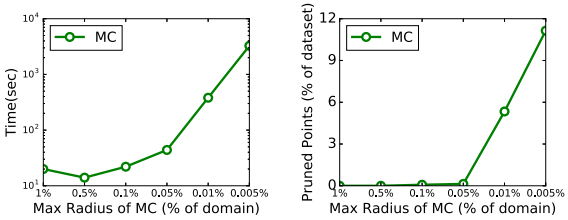
Pruning of MC. We also evaluate the pruning strategy of the state-of-the-art Top-n LOF algorithm *MC*. The OpenStreetMap Rhode Island dataset is utilized in this set of experiments. The parameter k (3) and n (1) are set much smaller than other experiments, because based on our testing, MC cannot prune any point when k and n are set to larger values. As shown in Fig. 7(b) as the radius of the micro-clusters (formed for pruning purpose) decreases, the percentage of the pruned points increases up to 11%. However, the processing time also increases accordingly, although more points are pruned. This is so because a smaller radius leads to the generation of a large number of small micro-clusters. Although small cluster benefits the pruning strategy of MC, generating a large number of clusters significantly increases the cost of Birch clustering when creating the CF tree [23]. The increased preprocessing costs outweigh the benefit of pruning more points. Therefore the pruning of MC only works in very limited scenarios with carefully tuned cluster radius parameter.

5.5 Evaluation of the Influence of Parameters

We next evaluate the influence of the number of neighbors k and the number of outliers n. We use the OpenStreetMap Connecticut dataset with 2.1 million records (Tab. 1).

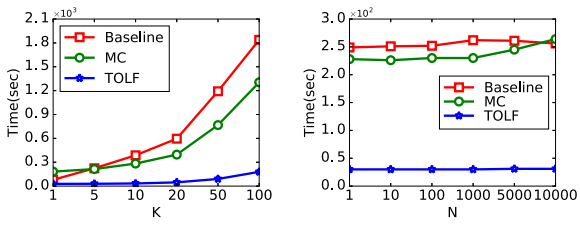
Influence of Varying Parameter k. Fig. 8(a) presents the results of varying the LOF input parameter k from 1 to 100. *D-TOLF* outperforms other alternatives up to 1 order of magnitude in total processing time even on this small dataset. As k increases, the costs of the kNN search will also increase and hence the overall processing time. However the processing time of D-TOLF increases much slower than *baseline* and *MC*. Therefore the larger k is, the more D-TOLF wins. This is so because the multi-granularity pruning strategy of D-TOLF effectively reduces the number of kNN searches required by the exact LOF score computation. Further, our density-aware forest index is more effective in speeding up the kNN search than the traditional indexing like R-tree [10] utilized by *baseline* and *MC*, while the kNN search becomes more expensive as k increases.

Influence of 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 1 to 10,000. *D-TOLF* beats *baseline* and *MC* in all cases at least one order of magnitude. Further, the processing time of D-TOLF is stable as n increases. This indicates that the pruning and indexing of D-TOLF are still very effective with large n. The processing time of *baseline* and *MC* increases slightly when n increases. For baseline, their additional sorting phase becomes more expensive as n increases. However, the costs of the sorting phase are minor compared to the LOF score computation costs. As for *MC* the increase of n potentially would influence its pruning ability. However, since the pruning ability of *MC* is already very limited even when n is small, the influence of a larger n to *MC* will not be very obvious.



(a) Time Cost (b) Percentage of Pruned Points

Figure 7: Evaluate the Pruning of MC on Rhode Island Dataset. (k=3, n=1)



(a) Tune K (b) Tune N

Figure 8: Tune Parameters k and n on Connecticut dataset

6 RELATED WORK

Breunig et al. [6] proposed the notion of local outliers in contrast to global distance-based outliers [15, 20]. 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 identifying anomalous points [16]. A centralized algorithm was proposed in [6] to compute the LOF score of each point. It can be utilized to detect top-n LOF outliers by sorting the points based their computed LOF scores. As a baseline method it is shown in our experiments to be at least 20 times slower than our TOLF approach, because it relies on routinely conducting the expensive kNN search on each point to detect outliers.

In [13] that proposed the concept of top-n LOF outlier, a centralized detection algorithm was developed that is highly coupled with an expensive preprocessing phase [13]. It first applies BIRCH clustering to group nearby data points together. Then based on the radius of each individual cluster and the distance relationships among all clusters, it ranks the clusters based on their likelihood of containing outliers and detects outliers only in the highly ranked clusters. However as shown in its experiments [13], this method took thousands of seconds to process a synthetic dataset smaller than 1M. Therefore it is not scalable to reasonable sized datasets. As confirmed by our experiments in Sec. 5.2, in some cases it cannot even beat the naive two step baseline approach due to its high preprocessing costs and inefficiency in avoiding the expensive exact LOF score computation.

Another local outlier detection technique was proposed in [19] called LOCI. Unlike LOF, LOCI utilizes a distance range threshold r to define the local neighborhood of each point instead of the kNN concept. LOCI has lower computation costs compared to LOF. However, applying a unified distance range threshold r to the whole dataset is not effective in defining the local neighborhood when handling skewed datasets. It may lead to a vacant neighborhood for data points in sparse areas, while numerous neighbors for data points in dense areas.

Bhaduri et al. [5] proposed an efficient detection method for the distance-based outlier semantics in [20] which defines outliers as the n points presenting the highest k-distance, where k-distance represents the distance of a point to its kth nearest neighbor. Similar to our work, it utilizes the nth largest k-distance seen so far as a threshold to determine whether a new point p has chance to be in the top-n list based on its largest possible (upper bound) k-distance. Given a point p estimating its upper bound for k-distance is much more straightforward than for LOF score. Intuitively, in the k-distance computation process, the distance of p to its kth furthest point evaluated so far (or temporary k-distance) can naturally serve as the upper bound k-distance of p. This is so because the more points are evaluated, the smaller the temporary k-distance will be. Unfortunately, the LOF score computation does not demonstrate such monotonicity property. Approximating the upper bound of LOF score is much more challenging than k-distance, since the LOF score of p is the ratio of its local density against the average local density of its kNN, which is determined not only by p's kNN, but also by its kNN's kNN.

7 CONCLUSION

Top-n Local Outlier Factor semantics (*LOF*) is shown to be very effective in detecting outliers in skewed real world large data. However existing techniques lack proper scaling to large dataset. In this work, we propose the first scalable Top-n *LOF* outlier detection approach called TOLF. Innovations include multi-granularity pruning strategy that quickly excludes most inliers without computing their LOF scores and even kNNs, a data-driven partitioning strategy that ensures the effectiveness of the pruning strategy over skewed data, and the density-aware forest indexing mechanism to speed up kNN search. Our experimental evaluation on OpenStreetMap, TIGER, and SDSS datasets demonstrates the efficiency of TOLF - up to 35 times faster than the state-of-the-art approach.

8 ACKNOWLEDGEMENT

This work is supported by NSF IIS #1560229, NSF CRI #1305258, and Philips Research.

REFERENCES

[1] 2016. SDSS 13 Release. http://www.sdss.org/dr13/. (2016).
[2] 2017. Top-n LOF Code. https://github.com/yizhouyan/TopNLOFKDD. (2017).
[3] Charu C. Aggarwal. 2013. *Outlier Analysis*. Springer.
[4] Stephen D. Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*. 29–38.
[5] Kanishka Bhaduri, Bryan L. Matthews, and Chris Giannella. 2011. Algorithms for Speeding up Distance-based Outlier Detection. In *KDD*. 859–867.
[6] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and JÃČÂűrg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD (SIGMOD '00)*. ACM, New York, NY, USA, 93–104.
[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms (3. ed.)*. MIT Press.
[8] Ahmed Eldawy and Mohamed F Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*. IEEE, 1352–1363.
[9] Kyle S. Dawson et.al. 2016. The SDSS-IV Extended Baryon Oscillation Spectroscopic Survey: Overview and Early Data. *The Astronomical Journal* 151, 2 (2016), 44.
[10] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 47–57.
[11] Mordechai Haklay and Patrick Weber. 2008. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing* 7, 4 (2008), 12–18.
[12] Douglas M. Hawkins. 1980. *Identification of Outliers*. Springer. 1–188 pages.
[13] Wen Jin, Anthony K. H. Tung, and Jiawei Han. 2001. Mining Top-n Local Outliers in Large Databases. In *KDD (KDD '01)*. New York, NY, USA, 293–298.
[14] Edwin M. Knorr and Raymond T. Ng. 1998. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *VLDB*. 392–403.
[15] Edwin M Knox and Raymond T Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*. 392–403.
[16] Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. 2003. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection.. In *SDM*. SIAM, 25–36.
[17] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. 2012. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1016–1027.
[18] Gustavo Henrique Orair, Carlos H. C. Teixeira, Ye Wang, Wagner Meira Jr., and Srinivasan Parthasarathy. 2010. Distance-Based Outlier Detection: Consolidation and Renewed Bearing. *PVLDB* 3, 2 (2010), 1469–1480.
[19] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. 2003. LOCI: Fast Outlier Detection Using the Local Correlation Integral. In *ICDE*. 315–326.
[20] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. In *SIGMOD*. 427–438.
[21] Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.* 16, 2 (June 1984), 187–260.
[22] Chi Zhang, Feifei Li, and Jeffrey Jestes. 2012. Efficient parallel kNN joins for large data in MapReduce. In *EDBT*. 38–49.
[23] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, Vol. 25. ACM, 103–114.