Fast and Scalable Outlier Detection with Approximate Nearest Neighbor Ensembles

Author manuscript – the final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-18123-3_2

Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel

Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany
http://www.dbs.ifi.lmu.de
{schube,zimek,kriegel}@dbs.ifi.lmu.de

Abstract. Popular outlier detection methods require the pairwise comparison of objects to compute the nearest neighbors. This inherently quadratic problem is not scalable to large data sets, making multidimensional outlier detection for big data still an open challenge. Existing approximate neighbor search methods are designed to preserve distances as well as possible. In this article, we present a highly scalable approach to compute the nearest neighbors of objects that instead focuses on preserving neighborhoods well using an ensemble of space-filling curves. We show that the method has near-linear complexity, can be distributed to clusters for computation, and preserves neighborhoods—but not distances—better than established methods such as locality sensitive hashing and projection indexed nearest neighbors. Furthermore, we demonstrate that, by preserving neighborhoods, the quality of outlier detection based on local density estimates is not only well retained but sometimes even improved, an effect that can be explained by relating our method to outlier detection ensembles. At the same time, the outlier detection process is accelerated by two orders of magnitude.

1 Introduction

Vast amounts of data require more and more refined data analysis techniques capable to process big data. While the volume of the data often decreases dramatically with selection, projection and aggregation, not all problems can be solved this way. The domain of outlier detection is a good example where individual records are of interest, not overall trends and frequent patterns. Summarization will lose the information of interest here and thus cannot be applied to outlier detection. In the data mining literature, a large variety of methods is based on object distances, assuming that outliers will essentially exhibit larger distances to their neighbors than inliers, i.e., the estimated local density is lower than the "usual" density level in the dataset. Without employing index structures, this requires the computation of all pairwise distances in the worst case.

Here we focus on methods for improving generically all such methods by fast approximations of the relevant neighbors. We demonstrate that the approximation error is negligible for the task of outlier detection. In the literature,

approximations of neighborhoods and distances have been used as a filter step. Here, we show that an approximate identification of the neighborhood is good enough for the common outlier detection methods since these do not actually require the neighbors *as such* but only as a means to derive a local *density estimate*. Notably, the detection accuracy can even improve by using an approximate neighborhood. We explain this effect and argue that a tendency to improve the accuracy of outlier detection by approximate neighborhood identification is not accidental but follows a certain bias, inherent to our method.

In the remainder, we will discuss outlier detection methods and their efficiency variants (Section 2). Then (Section 3), we reason about the theoretical background of our method and consequences for its performance, and introduce the core of our method. We demonstrate the effectiveness and efficiency in an extensive experimental analysis (Section 4) and conclude the paper with rules of thumb on the expected usefulness of the different methods (Section 5).

2 Related Work

2.1 Outlier Detection

Existing outlier detection methods differ in the way they model and find the outliers and, thus, in the assumptions they, implicitly or explicitly, rely on. The fundamentals for modern, database-oriented outlier detection methods (i.e., methods that are motivated by the need of being scalable to large data sets, where the exact meaning of "large" has changed over the years) have been laid in the statistics literature. A broader overview for modern applications has been presented by Chandola et al. [10]. Here, we focus on techniques based on computing distances (and derived secondary characteristics) in Euclidean data spaces.

With the first database-oriented approach, Knorr and Ng [22] triggered the data mining community to develop many different methods, typically with a focus on scalability. A method in the same spirit [33] uses the distances to the k nearest neighbors (kNN) of each object to rank the objects. A partition-based algorithm is then used to efficiently mine top-n outliers. As a variant, the sum of distances to all points within the set of k nearest neighbors (called the "weight") has been used as an outlier degree [4]. The so-called "density-based" approaches consider ratios between the local density around an object and the local density around its neighboring objects, starting with the seminal LOF [7] algorithm. Many variants adapted the original LOF idea in different aspects [36].

2.2 Approximate Neighborhoods

For approximate nearest neighbor search, the Johnson-Lindenstrauss lemma [19] states the existence and bounds of a projection of n objects into a lower dimensional space of dimensionality $\mathcal{O}(\log n/\varepsilon^2)$, such that the distances are preserved within a factor of $1+\varepsilon$. Matoušek [26] further improves these error bounds. The most interesting and surprising property is that the reduced dimensionality depends only logarithmically on the number of objects and on the error bound, but

not on the original dimensionality d. Different ways of obtaining such a projection have been proposed for common norms such as Manhattan and Euclidean distance. A popular choice are the "database-friendly" random projections [1], where 2/3 of the terms are 0 and the others ± 1 (along with a global scaling factor of $\sqrt{3}$), which can be computed more efficiently than the previously used matrices. Another popular choice are projections based on s-stable distributions [11], where the Cauchy distribution is known to be 1-stable and the Gaussian distribution to be 2-stable [44] (i.e., they preserve L_1 and L_2 norms well). An overview and empirical study on different variations of the Johnson-Lindenstrauss transform [38] indicates that a reduced dimensionality of $k = 2 \cdot \log n/\varepsilon^2$ will usually maintain the pairwise distances within the expected quality.

2.3 Outlier Detection with Approximate Neighborhoods

Wang et al. [39] propose outlier detection based on Locality Sensitive Hashing (LSH) [17, 14, 11]. However—in contrast to what the authors state—it cannot be used for "any distance-based outlier detection mechanism", but it will only be useful for global methods such as kNN-Outlier [33, 4]: the key idea of this method is to use LSH to identify low-density regions, and refine the objects in these regions first, as they are more likely to be in the top-n global outliers. For local outlier detection methods there may be interesting outliers within a globally dense region, though. As a consequence, the pruning rules this method relies upon will not be applicable. Projection-indexed nearest-neighbours (PINN) [12] shares the idea of using a random projection to reduce dimensionality. On the reduced dimensionality, a spatial index is then employed to find neighbor candidates that are refined to k nearest neighbors in the original data space.

Much research aimed at improving efficiency by algorithmic techniques, for example based on approximations or pruning techniques for mining the top-n outliers only [6, 29]. A broad and general analysis of efficiency techniques for outlier detection algorithms [30] identifies common principles or building blocks for efficient variants of the so-called "distance-based" models [22, 33, 4]. The most fundamental of these principles is "approximate nearest neighbor search" (ANNS). The use of this technique in the efficient variants studied by Orair et al. [30] is, however, different from the approach we are proposing here in a crucial point. Commonly, ANNS has been used as a filter step to discard objects from computing the *exact* outlier score. The exact kNN distance could only become smaller, not larger, in case some neighbor was missed by the approximation. Hence, if the upper bound of the kNN distance, coming along with the ANNS, is already too small to possibly qualify the considered point as a top-n outlier, the respective point will not be refined. For objects passing this filter step, the *exact neighborhood* is still required in order to compute the *exact outlier score*. All other efficiency techniques, as discussed by Orair et al. [30], are similarly based on this consideration and differ primarily in the pruning or ranking strategies. As opposed to using ANNS as a filter step, we argue to *directly* use approximate nearest neighbor search to compute outlier scores without this refinement i.e., we base the outlier score on the k *approximate* nearest neighbors directly.

2.4 Summary

The differentiation between "distance-based" and "density-based" approaches, commonly found in the literature, is somewhat arbitrary. For both families, the basic idea is to provide estimates of the density around some point. As opposed to statistical approaches, that fit specific distributions to the data, the density-estimates in the efficient database algorithms are, in a statistical sense, parameter-free, i.e., they do not assume a specific distribution but estimate the density-level, using typically some simple density model (such as the k nearest neighbor distance). This observation is crucial here. It justifies our technique also from the point of view of outlier detection models: the exact neighbors are usually not really important but just the estimate of the density-level around some point and the difference from estimated density-levels around other points. In most cases, this will derive just the same outliers as if the outlier detection model were based on the *exact* distances to the *exact* neighbors, since those outlier scores will always remain *just an estimate*, based on the data sample at hand, and not on the unknown true density-level. The same reasoning relates to ensemble methods for outlier detection [3, 40], where a better overall judgment is yielded by diversified models. Models are diversified using approximations of different kinds: subsets of features [23], subsets of the dataset [42], even by adding *noise* components to the data points in order to yield diverse density-estimates, the results for outlier detection ensembles can be improved [41]. As opposed to outlier detection ensemble methods, here, we push the ensemble principle of diversity and combination to a deeper level: instead of creating an ensemble from different outlier models, we create an ensemble of different neighborhood approximations and use the combined, ensemble approximation of the neighborhood as the base for the outlier model.

3 Efficient Outlier Detection

Outlier detection methods based on density estimates, such as the kNN [33] or weighted kNN [4] outlier models, as well as the Local Outlier Factor (LOF) [7] and its many variants [36], rely on the retrieval of nearest neighbors for each data object. While the major part of database research on efficient outlier detection focused on retrieving the exact values of the top n outliers as fast as possible, using approximate neighborhoods as a filter, we maintain here that outlier detection methods do not heavily rely on the neighborhood sets to be *exact*: they use the distance to the kNN to estimate a density, local methods additionally use the kNN to compute an average neighbor density as reference. As long as the distances are not influenced heavily by the approximation, and the reference (approximate) neighbors still have a similar density, the results are expected to be similar. For approximations of neighborhoods by space filling curves the approximation error is not symmetric: it will never underestimate a k-distance, but by missing some true nearest neighbors it will instead return the $k + e$-distance for $e \geq 0$. The difference between the k and the $k + e$ distance is expected to be rather small in dense areas (i.e., for "inliers"), as there are

many neighbors at similar distances. For an object in a sparse area (i.e., an "outlier"), the $k+1$-distance can already be much larger than the k-distance. We can expect, on average, the scores of true outliers to further increase, which is well acceptable for the purpose of detecting outliers.

In computer science and data analysis, we rely on mathematics for correctness of the results. Yet, we also have to deal with the fact that neither our computation will be perfect—due to limited numerical precision—nor our data are exact: even unlimited data will still only yield an approximation of reality. With our data only being a finite sample, chances are that the exact computation will not be substantially closer to the truth than a good approximation. Of course we must not give up precision without having benefits from doing so. However, for large data sets we have an immediate problem to solve: finding the nearest neighbors by computing a distance matrix, or by repeated linear scans, will not scale to such data sets anymore. Trading some precision for reducing the runtime from quadratic to linear may be well worth the effort.

3.1 Approximate Indexing Techniques

There exist capable indexing methods for low-dimensional data such as the k-d tree and the R*-tree. In order to use such techniques for high dimensional data, the data dimensionality must be reduced. Approaches in the context of outlier detection are feature bagging [23] and PINN [12], using Achlioptas' database-friendly random projections [1]. Locality Sensitive Hashing (LSH, [17, 14]) uses s-stable random projections [11] for indexing Minkowski distances and found use in outlier detection as well [39]. LSH (on dense vector data with L_p-norms) combines multiple established strategies:

1. dimensionality reduction by s-stable random projections to k dimensions;
2. grid-based data binning into $\mathbb{N}^d$ bins of width w;
3. reduction of grid size by hashing to a finite bin index;
4. similarity search in the bin of the query point only;
5. ensemble of ℓ such projections.

Individual parts of this approach can be substituted to accommodate different data types and similarity functions. For example, instead of computing the hash code on a regular grid, it can be based on the bit string of the raw data, or a bit string obtained by splitting the data space using random hyperplanes. LSH is an approximate nearest neighbor search algorithm both due to the use of random projections (which only approximately preserve distances) but also due to searching within the same bin as the query point only. The use of hash tables makes it easy to parallelize and distribute on a cluster.

Projection-indexed nearest-neighbours (PINN) [12] also uses random projection to reduce dimensionality. A spatial index is then employed to find neighbor candidates in the projected space:

1. dimensionality reduction using "database friendly" random projections;
2. build a spatial index (R*-tree, k-d tree) on the projected data;

3. retrieve the $c \cdot k$ nearest neighbors in the projection;
4. refine candidates to k nearest neighbors in original data space.

Due to the use of random projections, this method may also not return the true k nearest neighbors, but it has a high probability of retrieving the correct neighbors [12]. In contrast to LSH, it is also guaranteed to return the desired number of neighbors and thus to always provide enough data for density estimation and reference sets to be used in outlier detection. When a true nearest neighbor is not found, the false positives will still be spatially close to the query point, whereas with LSH they could be any data.

The type of random projections discussed here are not a general purpose technique: the Johnson-Lindenstrauss lemma only gives the existence of a random projection that preserves the distances, but we may need to choose different projections for different distance functions. The projections discussed here were for unweighted L_p-norm distances. Furthermore it should be noted, as pointed out by Kabán [20], that random projection methods are not suitable to defy the "concentration of distances"-aspect of the "curse of dimensionality" [43]: since, according to the Johnson-Lindenstrauss lemma, distances are preserved approximately, these projections will also preserve the distance concentration.

3.2 Space-Filling Curves

Space-filling curves are a classic mathematical method for dimensionality reduction [31, 15]. In contrast to random projections, by space-filling curves the data are always reduced to a single dimension. In fact, the earliest proposed space-filling curves, such as the Peano curve [31] and the Hilbert curve [15], were defined originally for the two dimensional plane and have only later been generalized to higher dimensionality. A space-filling curve is a fractal line in a bounded d dimensional space (usually $[0; 1]^d$) with a Hausdorff dimensionality of d that will actually pass through every point of the space.

The first curve used for databases was the Z-order. Morton [27] used it for indexing multidimensional data for range searching, hence the Z-order is also known as Morton code and Lebesgue curve. This curve can be obtained by interleaving the bits of two bit strings x_i and y_i into a new bit string: $(x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4 \ldots)$. The first mathematically analyzed space-filling curve was the Peano curve [31], closely followed by the Hilbert curve [15] which is considered to have the best mathematical properties. The Peano curve has not received much attention from the data indexing community because it splits the data space into thirds, which makes the encoding of coordinates complex. The Hilbert curve, while tricky in high dimensional data due to the different rotations of the primitives, can be implemented efficiently with bit operations [8], and has been used, e.g., for bulk-loading the R*-tree [21] and for image retrieval [28].

Indexing data with space-filling curves as suggested by Morton [27] is straightforward: the data are projected to the 1-dimensional coordinate, then indexed using a B-tree or similar data structure. However, *querying* such data is challenging: while this index can answer exact matches and rectangular window queries

well, finding the exact k nearest neighbors is nontrivial. Thus, for outlier detection, we will need query windows of different size in order to find the k nearest neighbors. A basic version of this method for high dimensional similarity search [37] used a large number of "randomly rotated and shifted" curves for image retrieval. A variant of this approach [25] uses multiple systematically shifted – not rotated – copies of Hilbert curves and gives an error bound based on Chan's work [9]. To retrieve the k nearest neighbors, both methods look at the preceding and succeeding k objects in each curve, and refine this set of candidates.

Chan [9] gave approximation guarantees for grid-based indexes based on shifting the data diagonally by $1/(d+1)$ times the data extent on each axis. The proof shows that a data point must be at least $1/(2d+2) \cdot 2^{-\ell}$ away from the nearest border of the surrounding cell of size $2^{-\ell}$ (for any level $\ell \geq 0$) in at least one of these curves due to the pigeonhole principle. Within at least one grid cell, all neighborhoods within a radius of $1/(2d+2) \cdot 2^{-\ell}$ therefore are in the same grid cell (i.e., nearby on the same curve). By looking at the length of the shared bit string prefix, we can easily determine the ℓ which we have fully explored, and then stop as desired. An approximate k-nearest neighbor search on such curves – by looking at the k predecessors and successors on each of the $d+1$ curves only – returns approximate k nearest neighbors which are at most $\mathcal{O}(d^{1+1/p})$ farther than the exact k nearest neighbors, for any L_p-norm [25]. For the 1^{st}-nearest neighbor, the error factor is at most $d^{1/p}(4d+4)+1$ [25].

In our approach, we divert from using the systematic diagonal shifting for which these error bounds are proved. It can be expected that the errors obtained by randomized projections are on a similar scale on *average*, but we cannot guarantee such bounds for the worst case anymore. We do however achieve better scalability due to the lower dimensionality of our projections, we gain the ability to use other space filling curves, and are not restricted to using $d+1$ curves. Similar to how diversity improves outlier ensembles [35, 40], we can expect diverse random subspaces to improve the detection result.

Space filling curves are easy to use in low dimensional space, but will not trivially scale up to high dimensionality due to the combinatorial explosion (just as any other grid based approach) [43]. They work on recursive subdivisioning of the data space, into 2^d (3^d for the Peano curve) cells, a number which grows exponentially with the dimensionality d. In most cases, the ordering of points will then be determined by binary splits on the first few dimensions only. HilOut [4] suffers both from this aspect of the curse of dimensionality, and from the distance concentration which reduces its capability to prune outlier candidates: since all distances are increasingly similar, the set of outlier candidates does not shrink much with each iteration of HilOut. For this top-n method to perform well, it must be able to shrink the set of candidates to a minimum fast, so that it can analyze a wider window of neighbors.

3.3 Fast Approximate kNN Search

Our proposed method to search for nearest neighbors is closely inspired by the methods discussed before, such as HilOut. However, it is designed with paral-

Algorithm 1: Phase 1: Projection and Data Rearragement

```
distributed on every node do                          // Project data locally
    foreach block do
        foreach curve do
            project data to curve
            store projected data
            send sample to coordination node
on coordination node do                 // Estimate distribution for sorting
    foreach curve do
        Read sample
        Sort sample
        Estimate global data distribution
        send global quantiles to every node
distributed on every node do                    // Rearrange data in cluster
    foreach curve do
        foreach projected block do
            split according to global quantiles
shuffle to new blocks
```

lelism and distributed computation in mind: where HilOut uses a nested loops approach to refine the current top candidates for a single outlier detection model, we focus on a method to compute the k nearest neighbors of all objects (often called kNN-self-join), so that we can then use an arbitrary kNN-based outlier detection method. At the same time, our method becomes easy to parallelize.

The principle of the proposed method is:

1. Generate m space-filling curves, by varying
   (a) curve families (Hilbert, Peano, Z-curve),
   (b) random projections and/or subspaces,
   (c) shift offsets to decorrelate discontinuities.
2. Project the data to each space-filling-curve.
3. Sort data on each space-filling-curve.
4. Using a sliding window of width $w \times k$, generate candidates for each point.
5. Merge the neighbor candidates across all curves and remove duplicates.
6. Compute the distance to each candidate, and keep the k nearest neighbors.

The parameter m controls the number of curves, and w can be used to control the tradeoff between recall and runtime, with $w = 1$ being a reasonable default. The proposed algorithm can be broken into three phases. We assume that the data are organized in blocks in a distributed file system such as HDFS (which provides built-in functionality for data chunking) or Sparks sliced RDDs.

In the first phase (Algorithm 1), the data are projected to each space-filling curve. The resulting data are stored on the local node, and only a sample is sent to the central node for estimating the data distribution. The central node then reads the projected samples, and estimates the global data distribution. The resulting split points are then distributed to each node, and the data are read a second time and reorganized into the desired partitions via the shuffle

Algorithm 2: Phase 2: Compute kNN and RkNN

```
distributed on every node do                          // Process sliding windows
    foreach curve do
        foreach projected, shuffled block do
            Sort block
            foreach object (using sliding windows) do
                emit (object, neighbors)
shuffle to (object, neighbor list)
distributed on every node do                  // compute kNN and build RkNN
    foreach (object, neighbor list) do
        Remove duplicates from neighbor list
        Compute distances
        emit (object, neighbors, ∅)                    // Keep forward neighbors
        foreach neighbor do
            emit (neighbor, ∅, [object])               // Build reverse neighbors
shuffle to (object, kNN, RkNN)
```

process in map-reduce. This sorting strategy was shown to scale to 100 TB in TritonSort [34]. While this is not a formal map-reduce process (requiring the transmission and use of auxiliary data), an implementation of this sorting process can be found in the Hadoop "terasort" example. The first phase serves as preprocessing to avoid having to project the data twice, and partially sorts the data according to the spatial curves. The required storage and communication cost is obviously $O(n \cdot m)$, i.e., linear in the data size and number of curves.

Algorithm 2 reads the output of the first phase. Each block in this data is a contiguous part of a space filling curve. We first finish the distributed sorting procedure within this data block. Then we can use a sliding window over the sorted data set to obtain neighbor candidates of each point. By emitting *(object, neighbor)* pairs to the map-reduce shuffle, we can easily reorganize the data to a *(object, neighbor list)* data layout and remove duplicates. For many local outlier detection algorithms, we will also need the reverse k-nearest neighbors (RkNN) to orchestrate model redistribution. This can be achieved by emitting inverted triples *(neighbor, ∅, object)*. The shuffle process will then reorganize the data such that for each object we have a triple *(object, neighbors, reverse neighbors)*.

In the third phase (Algorithm 3), we then compute the outlier scores using the generalized model of Schubert et al. [36]. In the experiments, we will use the LOF [7] model in this phase. Obviously, one can run other kNN-, SNN- [16], and reverse-kNN-based [18, 32] algorithms on the precomputed neighborhoods as well in the same framework. The reverse-kNNs are computed by simple list inversion to optimize data communication: this makes it easy to transmit an object's density estimate to the neighbor objects for comparison.

3.4 Favorable Bias of the Approximation

There exists an interesting bias in the approximation using space-filling curves (SFCs), which makes them particularly useful for outlier detection. The error

Algorithm 3: Phase 3: Compute Outlier Scores

```
distributed on every node do                                  // Compute models
    foreach (object, kNN, RkNN) do
        Compute model for object                               // Build own model
        emit (object, (object, model))                         // Retain own model
        emit (reverse neighbor, (object, model))               // Distribute model
shuffle to (object, model list)                                // Collect models
distributed on every node do                                   // Compare models
    foreach (object, (neighbor, model)) do
        Compare model to neighbor models
        Store outlier score for model
        Collect outlier score statistics                       // (for normalization)
    emit Send statistics to coordination node
on coordination node do                                        // Normalize Outlier Scores
    Merge outlier score statistics
    send statistics to every node
distributed on every node do
    foreach (object, score) do
        Normalize Outlier Score
        if score above threshold then
            emit (outlier, normalized score)
```

introduced by SFCs scales with the density of the data: if the bit strings of two vectors agree on the first $d \cdot \ell$ bits, the vectors are approximately within a cube of edge length $2^{-\ell}$ times the original data space.

For query points in a dense region of the data, the explored neighbors will be closely nearby, whereas for objects in less dense areas (i.e., outliers) the error introduced this way will be much larger on average. Furthermore, for an object central to a cluster, "wrong" nearest neighbors tend to be still members of the same cluster, and will just be slightly farther away.

For an outlier however, missing one of the true nearest neighbors – which may be another outlier with low density – and instead taking an even farther object as neighbor actually increases the chance that we end up using a cluster member of a nearby cluster for comparison. So while the approximation will likely not affect inlier scores much, we can expect it to emphasize outliers.

This effect is related to an observation for subsampling ensembles for outlier detection [42]: when subsampling a relative share of s objects from a uniformly distributed ball, the kNN-distances are expected to increase by a relative factor of $(1 - s^{1/d})/s^{1/d}$. Since for outliers this distance is expected to be higher, the expected increase will also be larger, and thus the outlier will become more pronounced with respect to this measure.

For other use cases such as density based cluster analysis, the effects of this approximation may be much more problematic. Such methods may fail to discover connected components correctly when a cluster is cut into half by a discontinuity in the space-filling curve: in contrast to outlier detection which only requires representative neighbors, such methods may rely on complete neighbors.

3.5 Discussion

Though our approach and the related approaches, PINN [12] and LSH-based outlier detection [39], can all be used to find the approximate nearest neighbors efficiently, they are based on subtly different foundations of approximation.

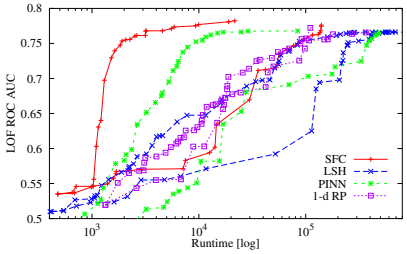
Random projections are designed to approximately *preserve distances*, while reducing dimensionality. Using an exact index on the projected data, as done in PINN, will therefore find the k nearest neighbors with respect to the approximate distance. The index based on locality sensitive hashing (LSH) in contrary is lossy: it is designed to have a high chance of *preserving regions of a fixed size* w, where the size w is a critical input parameter: the smaller the size that needs to be preserved, the faster the index; when the parameter is chosen too high, all objects will be hashed into the same bin, and the index will degenerate to a linear scan.

Space-filling curves on the contrary neither aim at directly preserving distances, nor do they try to preserve regions of a given radius. Instead, space-filling curves try to *preserve closeness*: the nearest neighbors of an object will often be nearby on the curve, while far neighbors in the data space will often be far away on the curve as well. For the purpose of density-based outlier detection, this yields an important effect: the index based on space-filling curves is better at adapting to different densities in the data set than the other two indexes, which makes it more appropriate for local density-based outlier detection methods.

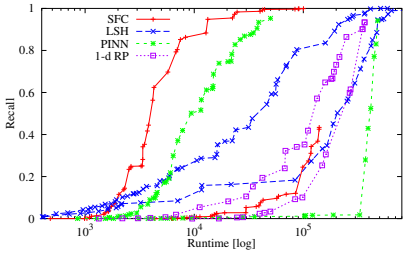
4 Experiments

For the experiments, all methods were implemented in ELKI [2]. We tested the behavior of our method as well as the related approaches PINN [12] and LSH-based outlier detection on several datasets. As a reference, we have also LOF results based on an exact index, using the R*-tree in different variants (i.e., different page sizes, different bulkload strategies).

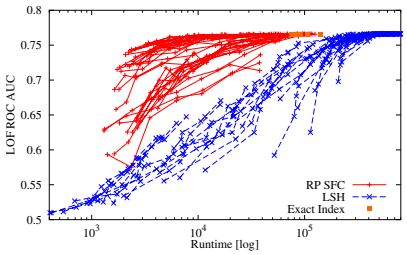
In our experiments, we used a number of larger data sets. From the image database ALOI [13], containing 110,250 images, we extracted 27 and 64 dimensional color histogram vectors. In order to obtain an outlier data set, a random subset of the classes were downsampled, so that the final data set contains only 75,000 images, of which 717 are labeled as outliers [16]. We also extracted all geographic coordinates from DBpedia [24] (Wikipedia preprocessed as RDF triples). This 2-dimensional data set does not have labeled outliers, and thus we used the top-1% according to LOF as outliers. This allows to see how close methods based on approximate neighborhoods come to the exact LOF results. However, as it is a low dimensional dataset, runtime results demonstrate that R*-tree indexes can work well. The forest covertype data set from the UCI machine learning repository [5] is a well known classification data set. The type cottonwood/willow is used as outlier class. In the following, we analyse the results on the 27 dimensional ALOI dataset in detail, as this data set has labeled outliers and is only of medium dimensionality. For the other datasets, we can draw similar conclusions and show only some sample results.



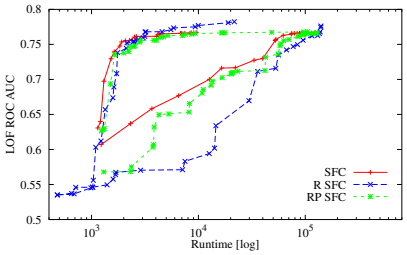
(a) Skyline results by basic method



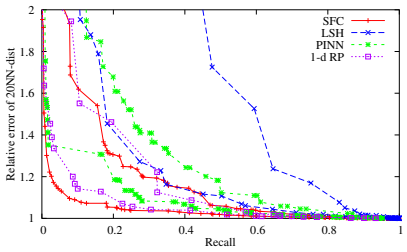
(b) Skyline results for recall of true 20 NN



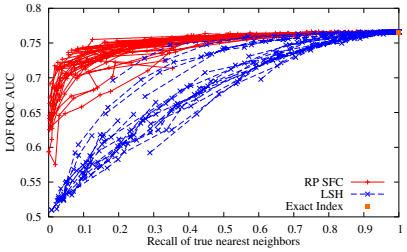
(c) Sample of results: random subspace SFC vs. LSH



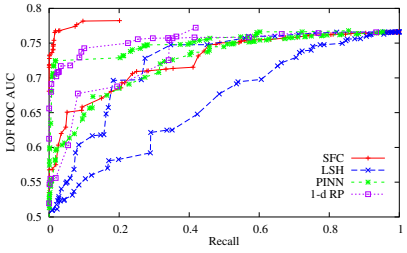
(d) Skyline results by SFC projection



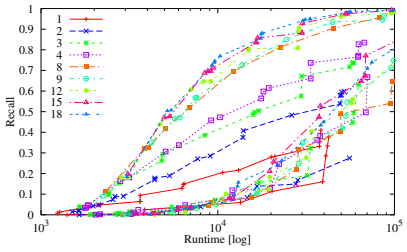
(e) Relative 20-dist error, compared to recall.



(f) Sample of results: Recall of 20NN vs. ROC AUC.



(g) Skyline results for ROC AUC vs. recall



(h) Ensemble effect combining multiple SFC

Fig. 1: Experiments on 27d ALOI data set. *SFC*: Space filling curves (proposed method), *R SFC*: Random features + SFC, *RP SFC*: Random projections + SFC, *LSH*: Locality sensitive hashing, *PINN*: Projection indexed nearest neighbors, *1-d RP*: one-dimensional random projections. *Exact index*: R*-tree

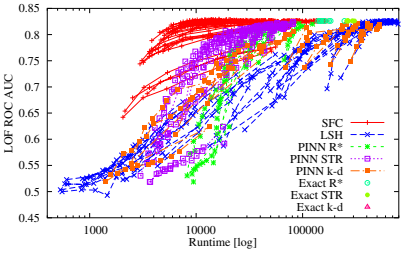
Figure 1 visualizes the results for running the LOF algorithm on the ALOI data set with $k = 20$ on a single CPU core. In total, we evaluated over 4,000 different index variations for this data set. To make the results readable we visualize only the skyline results in Figure 1a. The skyline are all objects where no other result is both faster and has a higher score at the same time (upper skyline) or where no other result is both slower and scores less at the same time (lower skyline). The upper skyline is useful for judging the *potential* of a method, when all parameters were chosen optimal, whereas the lower skyline indicates the worst case. In Figure 1c, we give a sample of the full parameter space we explored. Obviously, all 4,000 runs will be an unreadable cloud of points, thus we filter the results to LSH and only one variant of SFC (random curve families and random subspaces), which are about 700 runs. To show the continuity of the explored parameter space, we connected similar parametrizations with a line, more specifically for SFC indexes we connected those that differ only in window width w and for LSH we connect those that vary the number of hash tables ℓ. For exact indexes, different results arise for different page sizes. Note, though, that the skylines used typically represent hundreds of experiments. The skylines for LSH in Figure 1a represent the same set of results as in Figure 1c whereas the results for SFC in Figure 1c are restricted to the random variant and the skylines for SFC in Figure 1a include the other variants. Based on the systematic exploration of the parameter space as sampled in Figure 1c, it is interesting to see that with SFC we repeatedly were able to get higher outlier detection quality at a more than 10-fold speedup over the exact indexes. Merely when using a *single* space-filling curve, the results were not substantially better. Figure 1a visualizes the skyline of outlier detection quality vs. runtime, whereas Figure 1b is an evaluation of the actual index by measuring the recall of the true 20 nearest neighbors. In both measures, the SFC based method has the best potential – it can even be better than the exact indexes – while at the same time it is usually an order of magnitude faster than PINN and two orders of magnitude faster than LSH (both at comparable quality). Even if the parameters are chosen badly (which for SFC usually means using too few curves), the results are still comparable to LSH and PINN. However, there is a surprising difference between these two charts. They are using the same indexes, but the LOF ROC AUC scores for the SFC index start improving quickly at a runtime of 1,000-2,000 ms. The recall however, starts rising much slower, in the range of 1,500-10,000 ms. When we choose an average performing combination for the SFC index, e.g., 8 curves of different families combined with a random subspace projection to 8 dimensions and a window width of $w = 1$, we get a runtime of 4,989 ms, a ROC AUC of 0.747, and an average recall of the true nearest neighbors of 0.132. For an explanation for such a good performance *despite* the low recall, we refer the reader back to the reasoning provided in Section 3.4. In Figure 1d, we explore the effect of different random projections. The skyline, marked as "SFC", does not use random projections at all. The curves titled "R SFC" are space filling curves on randomly selected features only (i.e., feature bagging), while "RP SFC" uses full Achlioptas style random projections. As expected, the

variant without projections is fastest due to the lower computational cost. Using randomly selected features has the highest potential gains and in general the largest variance. Achlioptas random projections offer a similar performance as the full-dimensional SFC, but come at the extra cost of having to project the data, which makes them usually slower. Figure 1e visualizes the relative error of the 20-nearest neighbor distance over the recall. The SFC curves, despite a very low recall of less than 0.2, often suffer much smaller relative error than the other approaches. While the method does make *more* errors, the errors are *less severe*, i.e., the incorrect nearest neighbors have a smaller distance than those retrieved by the other methods. This is again evidence for the outlier-friendly bias of space filling curves (Section 3.4). Figure 1f is the same sample as Figure 1c, but projected to LOF ROC AUC quality and recall. One would naïvely expect that a low recall implies that the method cannot work well. While algorithm performance and recall are correlated for locality sensitive hashing, the SFC approach violates this intuition: even with very low recall, it already works surprisingly well; some of the best results have a recall of only around 0.25 – and outperform the exact solution. The bias (Section 3.4) again proves to be positive. When looking at the skylines of the complete data in Figure 1g, this even yields an upper skyline that ends at a recall of 0.2 – no result with a higher recall performed better than this. As particular 1-dimensional projections, space filling curves are by design more apt for low dimensional data rather than for high dimensional data. However, by combining multiple curves, i.e., building an ensemble of approximate neighborhood predictors, the performance gain is quite impressive also for high dimensional data. We show skylines for different numbers of curves combined in Figure 1h. While single curves are performing badly and remain unstable, combinations, here of up to 18 curves, improve considerably.

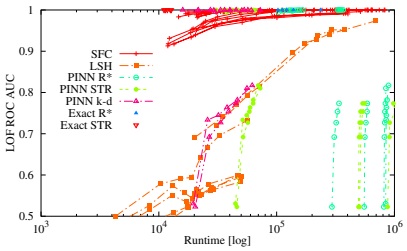
For other datasets, we show samples in Figure 2. For the 64 dimensional variant of ALOI (Figure 2a), including all variants of the exact indexes, backing PINN and backing LOF (i.e., "exact"), we can draw the same conclusions as for the 27d ALOI dataset. Again, our method is performing typically better at a shorter runtime. On the 2-dimensional DBpedia dataset (Figure 2b), as expected, we cannot beat the exact indices. However, in comparison with the other approximate methods, our method is performing excellent. For the large forest covertype dataset, let us study another aspect than those discussed before. We see in Figure 2c a reason for the faster runtime behavior of our method: we reach a good recall with far less distance computations than PINN or LSH.

5 Conclusion

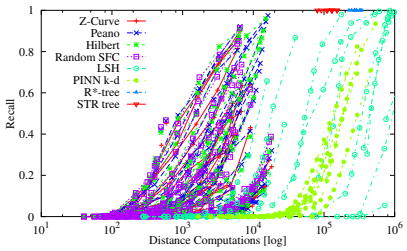
We proposed a method, based on space filling curves (that can be used in combination with other approximation methods such as random projections or LSH), for highly scalable outlier detection based on ensembles for approximate nearest neighbor search. As opposed to competing methods, our method can be easily distributed for parallel computing. We are hereby not only filling the gap of possible approximation techniques for outlier detection between random projections



(a) Results for ROC AUC vs. Runtime on 64d ALOI



(b) Results for ROC AUC vs. Runtime on DBpedia



(c) Results for Recall vs. Dist. Calc. on Covertype

Fig. 2: Sample results on other datasets

and LSH. We also show that this particular technique is more apt for outlier detection. Its competitive or even improved effectiveness is explained by a bias of space-filling curves favourable for outlier detection. Furthermore, the principle of combining different approximations is related to ensemble approaches for outlier detection [40], where the diversity is created at a different level than usual.

We rely on the same motivation as outlier detection ensembles: diversity is more important than getting the single most accurate outlier score because the exact outlier score of *some* method is not more than just some variant of the *estimate* of the density-level around some point and the difference from *estimated* density-levels around other points. Therefore, an approximate density estimate, based on approximate neighborhoods, will be typically good enough to identify just the same outliers as when computing the *exact* distances to the *exact* neighbors which will still be *only an estimate*, based on the data sample at hand, of the *true* but *inevitably unknown* density-level. Often, the results based on approximate neighborhoods are even better.

Based on our reasoning and experimental findings, we conclude with the following rules of thumb for the practitioner:

1. If the data dimensionality is low, bulk-loaded R*-trees are excellent.
2. If the exact distances are of importance, PINN is expected to work best.
3. If neighborhoods for a known, small radius w are needed, LSH is expected to work best.
4. If k-nearest neighborhoods are needed, as it is the case for the most well-known outlier detection methods, SFC is the method of choice.

References

1. Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. JCSS 66, 671–687 (2003)
2. Achtert, E., Kriegel, H.P., Schubert, E., Zimek, A.: Interactive data mining with 3D-Parallel-Coordinate-Trees. In: Proc. SIGMOD. pp. 1009–1012 (2013)
3. Aggarwal, C.C.: Outlier ensembles. SIGKDD Explor. 14(2), 49–58 (2012)
4. Angiulli, F., Pizzuti, C.: Outlier mining in large high-dimensional data sets. IEEE TKDE 17(2), 203–215 (2005)
5. Bache, K., Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
6. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proc. KDD. pp. 29–38 (2003)
7. Breunig, M.M., Kriegel, H.P., Ng, R., Sander, J.: LOF: Identifying density-based local outliers. In: Proc. SIGMOD. pp. 93–104 (2000)
8. Butz, A.R.: Alternative algorithm for Hilbert's space-filling curve. IEEE TC 100(4), 424–426 (1971)
9. Chan, T.M.: Approximate nearest neighbor queries revisited. Disc. & Comp. Geom. 20(3), 359–373 (1998)
10. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM CSUR 41(3), Article 15, 1–58 (2009)
11. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. ACM SoCG. pp. 253–262 (2004)
12. de Vries, T., Chawla, S., Houle, M.E.: Density-preserving projections for large-scale local anomaly detection. KAIS 32(1), 25–52 (2012)
13. Geusebroek, J.M., Burghouts, G.J., Smeulders, A.W.M.: The Amsterdam Library of Object Images. Int. J. Computer Vision 61(1), 103–112 (2005)
14. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proc. VLDB. pp. 518–529 (1999)
15. Hilbert, D.: Ueber die stetige Abbildung einer Linie auf ein Flächenstück. Math. Ann. 38(3), 459–460 (1891)
16. Houle, M.E., Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A.: Can shared-neighbor distances defeat the curse of dimensionality? In: Proc. SSDBM. pp. 482–500 (2010)
17. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc. STOC. pp. 604–613 (1998)
18. Jin, W., Tung, A.K.H., Han, J., Wang, W.: Ranking outliers using symmetric neighborhood relationship. In: Proc. PAKDD. pp. 577–593 (2006)
19. Johnson, W.B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. In: Conference in Modern Analysis and Probability, Contemporary Mathematics, vol. 26, pp. 189–206. American Mathematical Society (1984)
20. Kabán, A.: On the distance concentration awareness of certain data reduction techniques. Pattern Recognition 44(2), 265–277 (2011)
21. Kamel, I., Faloutsos, C.: Hilbert R-tree: An improved R-tree using fractals. In: Proc. VLDB. pp. 500–509 (1994)
22. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: Proc. VLDB. pp. 392–403 (1998)
23. Lazarevic, A., Kumar, V.: Feature bagging for outlier detection. In: Proc. KDD. pp. 157–166 (2005)

24. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web J. (2014)
25. Liao, S., Lopez, M.A., Leutenegger, S.T.: High dimensional similarity search with space filling curves. In: Proc. ICDE. pp. 615–622 (2001)
26. Matoušek, J.: On variants of the Johnson–Lindenstrauss lemma. Random Structures & Algorithms 33(2), 142–156 (2008)
27. Morton, G.M.: A computer oriented geodetic data base and a new technique in file sequencing. Tech. rep., International Business Machines Co. (1966)
28. Nguyen, G., Franco, P., Mullot, R., Ogier, J.M.: Mapping high dimensional features onto Hilbert curve: Applying to fast image retrieval. In: ICPR12. pp. 425–428 (2012)
29. Nguyen, H.V., Gopalkrishnan, V.: Efficient pruning schemes for distance-based outlier detection. In: Proc. ECML PKDD. pp. 160–175 (2009)
30. Orair, G.H., Teixeira, C., Wang, Y., Meira Jr., W., Parthasarathy, S.: Distance-based outlier detection: Consolidation and renewed bearing. PVLDB 3(2), 1469–1480 (2010)
31. Peano, G.: Sur une courbe, qui remplit toute une aire plane. Math. Ann. 36(1), 157–160 (1890)
32. Radovanović, M., Nanopoulos, A., Ivanović, M.: Reverse nearest neighbors in unsupervised distance-based outlier detection. IEEE TKDE (2014)
33. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: Proc. SIGMOD. pp. 427–438 (2000)
34. Rasmussen, A., Porter, G., Conley, M., Madhyastha, H., Mysore, R., Pucher, A., Vahdat, A.: TritonSort: a balanced large-scale sorting system. In: Proceedings of the 8th USENIX conference on Networked systems design and implementation (2011)
35. Schubert, E., Wojdanowski, R., Zimek, A., Kriegel, H.P.: On evaluation of outlier rankings and outlier scores. In: Proc. SDM. pp. 1047–1058 (2012)
36. Schubert, E., Zimek, A., Kriegel, H.P.: Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. Data Min. Knowl. Disc. 28(1), 190–237 (2014)
37. Shepherd, J.A., Zhu, X., Megiddo, N.: Fast indexing method for multidimensional nearest-neighbor search. In: Proc. SPIE. pp. 350–355 (1998)
38. Venkatasubramanian, S., Wang, Q.: The Johnson-Lindenstrauss transform: An empirical study. In: Proc. ALENEX Workshop (SIAM). pp. 164–173 (2011)
39. Wang, Y., Parthasarathy, S., Tatikonda, S.: Locality sensitive outlier detection: A ranking driven approach. In: Proc. ICDE. pp. 410–421 (2011)
40. Zimek, A., Campello, R.J.G.B., Sander, J.: Ensembles for unsupervised outlier detection: Challenges and research questions. SIGKDD Explor. 15(1), 11–22 (2013)
41. Zimek, A., Campello, R.J.G.B., Sander, J.: Data perturbation for outlier detection ensembles. In: Proc. SSDBM. pp. 13:1–12 (2014)
42. Zimek, A., Gaudet, M., Campello, R.J.G.B., Sander, J.: Subsampling for efficient and effective unsupervised outlier detection ensembles. In: Proc. KDD. pp. 428–436 (2013)
43. Zimek, A., Schubert, E., Kriegel, H.P.: A survey on unsupervised outlier detection in high-dimensional numerical data. Stat. Anal. Data Min. 5(5), 363–387 (2012)
44. Zolotarev, V.M.: One-dimensional stable distributions, Translations of Mathematical Monographs, vol. 65. American Mathematical Society (1986)