Abstract—k-d trees are one of the most common data structures used in the optimization problem of Nearest Neighbour Search (NNS), which to this day has issues both theoretical and practical concerning the curse of dimensionality. The article “High-dimensional approximate nearest neighbor: k-d Generalized Randomized Forests” presents a new data structure called generalized randomized k-d forest, k-d GeRaF that offers considerable competition to the current state-of-the-art algorithms for NNS. The authors prove that their method works efficiently on both synthetic and real datasets of a wide range of dimensions and sizes.

I. INTRODUCTION

Nearest Neighbour Search is an optimization problem that is found in many areas of application, for example machine learning, database query retrieval, pattern recognition, etc. Unfortunately, such problems often grow to extremely large scales: to hundreds of dimensions and more than a million data points, which makes it hard to balance between space and time efficiency in search algorithms. To rectify the curse of dimensionality, solutions often resort to approximate search. The authors propose a new method, which uses k-d trees, and is effective for datasets with dimensions between $10^3$ and $10^4$ and data points up to a million.

A. Previous work

For more than two decades, there have been attempts to construct a good approximation of NNS and currently there are many efficient solutions for it. One of the more competitive ones is the Balanced Box Decomposition (BBD) tree, a variant of the quadtree that subdivides space into hyper-rectangles aligned with the axes. Its space usage is $O(dbn)$, preprocessing time $O(dn \log n)$ and query time $O(d^{d+1} \log n / \epsilon^d)$. Another effective solution is from the library FLANN, which works best with dimensions around 100. FLANN uses a forest consisting up to 6 randomized k-d trees and performs simultaneous search in all of them. Coordinates that split the dataset are chosen adaptively but all leaves consist of a single data point. A method that specializes on high dimensional data is Locality Sensitive Hashing (LSH), which hashes the points of the dataset to ensure that collisions occur more likely for objects that are close to each other than for those, which are much different. LSH’s space requirement is about $O(dn^{1+\rho})$ and query time $O(dn^\rho)$ for some $\rho \in (0, 1)$.

II. THE K-D GERAFT

A. Randomization

The main principle of this method is to use randomized algorithms to construct considerably different trees. The authors tested four randomization factors, which they used either independently or in combination.

1) Rotation: For each k-d tree they randomly applied a different isometry, function to the point set, which transforms some data so that the distance between any two points does not change after the transformation. This, however, changes the dimensions for each tree. The trees are traversed using the transformation matrix $R$ on the query but the distances are calculated between the original, non-transformed values.

2) Split dimension: Instead of splitting using every dimension in order, the authors selected the $t$ dimensions of the highest variance for the dataset and chose one of them randomly for each node.

3) Split value: As the split value, the authors used the sum of the median value and a variable $\delta$ which is uniformly distributed in $[-3\Delta/\delta, 3\Delta/\delta]$ where $\Delta$ is the maximum Euclidean distance between any two points of the point set [2].

4) Shuffling: At the beginning of building the trees, the point sets for each tree are randomly shuffled, and thus different splits occur with each build.

B. Building the trees

At first, the $t$ dimensions of maximum variance are computed. To build a tree recursively, $X$ is transformed according to a distinct random function $f^*$, which applies an isometry or shuffling or both to the data. For each node one of the $t$ dimensions is chosen randomly and the dataset is split across the median. The child nodes are then built recursively in the same manner. Building stops when there are fewer points in the dataset than a certain $p$, which denotes the maximum number of points to store in a leaf node.

The algorithm’s complexity is $O(n)$ or $O(d)$, depending on whether the cardinality of the dataset ($n$) is much larger than the number of dimensions ($d$) or the opposite. The required space in total is $O(n(d + m))$, as $O(nm)$ is for the trees ($m$ being size of the forest) and $O(nd)$ is for the entire data structure.
C. Querying

The nodes are searched in all trees in parallel, using a shared min-priority queue $Q$. The tree is traversed, taking the most promising nodes from all trees via $Q$. The importance criterion is the distance between the query and the hyperplane that a node defines. When $Q$ is empty or the number of checked leaves reaches a certain boundary $c/(1 + \epsilon)$, where $\epsilon$ is an accuracy measure between 0 and 1, then the search is stopped. At first, all of the leaves in forest $F$ are traversed without checking any leaves and storing visited nodes in $Q$. Then, nodes are extracted from $Q$ and the trees are traversed accordingly, but the distances to all points in the leaves are now calculated. For each decision made between child nodes while traversing, one is visited and the other one is stored in $Q$. The neighbouring points are stored in a min-heap $H$ which is dynamically updated to contain maximum $k$ points, where $k$ is the number of neighbours to be returned. There is a separate array for keeping track of points that have already been visited to avoid unnecessary distance computations.

III. EXPERIMENT RESULTS

The authors used five datasets, two of which were synthetic (Klein bottle and Sphere), while the three other were common machine learning datasets MNIST (784×60k), SIFT (128×10⁶) and GIST (960×10⁶). Every experiment was conducted on a processor at 2.40 GHz×4 with 3.8 GB memory, except for the GIST dataset, for which the authors used a processor at 3 GHz×4 with 8 GB. The performance of GeoRaF was compared with BBD-trees, LSH, and FLANN. As the authors measured the accuracy using only the first result, which is why the two randomization factors were disabled for further experiments.

A. Preprocessing

Preprocessing includes building for each method, but for FLANN and GeoRaF also automatic parameter configuration. From the results in Figure 1, we can observe that GeoRaF is almost always faster than the other methods by one order of magnitude, with FLANN being quite slow and unable to configure parameters under 4hr for the Klein bottle dataset.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
          & Sphere $n = 10^6$, $d = 10^3$ & Klein $n = 10^6$, $d = 10^3$ & MNIST $n = 60k$, $d = 784$ & \hline
\hline
BBB       & 1.25 & 1.26 & 1.29 & 1.20 & 0.13 & 0.14 & 0.17 & 0.14 & 187.5 & 184.3 & 185.1 & 186.0 & \\
LSH       & 0.21 & 0.16 & 0.16 & 0.31 & 0.11 & 0.07 & 0.03 & 0.05 & 1.47 & 69.76 & 48.47 & 14.35 & \\
FLANN     & 25.0 & 25.4 & 25.5 & 25.6 & - & - & - & - & 244.6 & 217.2 & 157.3 & 142.0 & \\
GeoRaF    & 0.06 & 0.06 & 0.06 & 0.06 & 0.06 & 0.06 & 0.06 & 0.08 & 8.167 & 8.567 & 8.579 & 8.565 & \\
\hline
\end{tabular}
\caption{Results for preprocessing. $\epsilon$ is a precision parameter related to building.}
\end{table}

The building time for GeoRaF may increase with $\epsilon$ since it decreases the number of points per leaf and thus, yields more subdivisions.

B. Search

Figure 2 presents results for querying the synthetic datasets.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{search_results.png}
\caption{Search results. The miss rate refers to the percentage of queries where the reported neighbour was not correct.}
\end{figure}

GeoRaF mostly dominates the other methods in both speed and miss rate, except for FLANN on Sphere in the case of $d = 10^4$. BBD and FLANN have quite high miss rates, while LSH has at least 10 times slower than GeoRaF.

With real data (MNIST, GIST, SIFT), BBD and FLANN failed with automatic build or memory overload. Using manual parameter setting however, FLANN is the fastest method with poor accuracies. Generally, LSH has the same accuracy as GeoRaF but loses in speed by at least an order of magnitude.

The authors also measured for GeoRaF the percentage of queries where the reported nearest neighbour was out of the $1 + \epsilon$ bounds from the nearest distance. For Klein bottle with $n = 10^4$ and $d = 10^2$ this was determined to be 2% and 0% for $\epsilon = 0$ and $\epsilon \in \{0.1, 0.5, 0.9\}$ respectively. For GIST with $n = 10^5$ and $d = 960$, the percentages were 0% and 0.4% with $\epsilon = 0$ and $\epsilon \in \{0.1, 0.5, 0.9\}$.

IV. CONCLUSION

GeoRaF is an efficient data structure for approximate NNS that uses randomization techniques and is able to compete with current state-of-the-art algorithms. Experiments showed that it yields a good trade-off between speed and accuracy compared to the other methods, where some were unstable with even building the data structures. The article thoroughly presents its proofs and experiment results and is itself an interesting, though a quite difficult read when it comes to some technical details. Nevertheless, its competence in ANNS research is solid and, as it is unclear if GeoRaF supports insertions and deletions, it also leaves an open question for future research.

ACKNOWLEDGMENT

The author of this report has received the IT Academy Specialization Stipend for the Autumn semester of academic year 2017/18.

REFERENCES