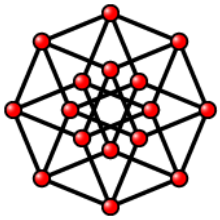
REasoning for COmplex Data

Institute of Computing

University of Campinas



Doctoral Research Project

Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search

Candidate: Javier Alvaro Vargas Muñoz
javier.munoz@ic.unicamp.br
Advisor: Prof. Dr. Ricardo da Silva Torres
rtorres@ic.unicamp.br
Co-Advisor: Prof. Dr. Zanoni Dias
zanoni@ic.unicamp.br

Abstract

A widely-used approach to represent multimedia objects to give support to recognition, classification, and retrieval tasks, relies on the use of feature vectors. Usually, the most accurate methods use high dimensional vectors to represent each object and work over large collections of objects. The search for the most similar objects in those collections is known as the *nearest neighbors search* problem, and is a key component in many tasks. The naïve approach for this problem consists in comparing the query object with all the objects of the collection and then sorting them according to the similarity (known as *linear search*). However, it becomes impractical when working over large collections and a high number of queries. Alternative exact methods, such as space-partitioning-trees based techniques, are very efficient with low dimensional data, but quickly converge to linear search when dimensionality increases. Approximate methods have shown considerable gains in efficiency in these kinds of scenarios with small losses in precision. In this proposal, we are interested in investigating methods for the *approximate nearest neighbors search* problem, specially on those techniques that create graphs of nearest neighbors and then perform the search over the graph in a greedy manner at query time. We propose a novel clustering-based graph construction approach, where we plan to exploit the proximity information between objects in the same cluster to create the edges in the graph. Also, we want to investigate novel search approaches to guide the navigation on the graph without computing exhaustively the distances to all neighbors in each step of the search, just through those in the *direction* of the query. We intend to evaluate the proposed approach in large datasets (in the scale of millions) of high dimensional data (e.g., SIFT, GIST, and CNN features) as those used in recent works and benchmarks for approximate nearest neighbor search.

1 Introduction

Nearest neighbors search is a broadly component used in many Information Retrieval, Computer Vision, and Machine learning tasks. Therefore, its fast execution is of paramount importance, especially when we are working with large collections of high dimensional data. Data structures for exact nearest neighbors (NN) search [1, 2] are very efficient when data dimensionality is low, but they suffer dramatically with the course of dimensionality when the data dimension increases. Approximate nearest neighbors search techniques are suitable when they yield acceptable results with a small loss of precision, in return of a fast search. In this proposal, we focus on Approximate Nearest Neighbors Search (ANNS).

There are two well-known variants of the nearest neighbors search problem: R-nearest-neighbors and K-nearest-neighbors. The first consist in search all the points inside of a hypersphere of radius R and center a query point q. The second is related to the search of the K nearest points to a query, in all the collection of points. This proposal is focused in the last problem. Formalizing, given a set of points $P = \{p_1, p_2, p_3, ..., p_n\}$, $P \subset \mathbb{R}^d$, and a distance function $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, the set of K-Nearest Neighbors to a query point q, is defined by $KNN(q, K, P) = A$, where A is set with the properties $|A| = K, A \subseteq P$, and $\forall x \in A, y \in P - A,\ D(q, x) < D(q, y)$.

The three most employed approaches for ANNS are based on tree indexing schemes, hash-based solutions, and neighborhood graphs. Tree partitioning structures present a natural way to organize the data. At each tree's level, the objects are split into subsets based on some criteria, which is then recursively applied to each subset until some stop condition is reached. Search is performed traversing from the root to the leaves. In most of these methods, search is performed using multiple trees. Usually these techniques present cheap costs for index construction. Within these category, we can mention the methods: KDTree [1], Hierarchical K-Means [3], FLANN [4], and VPTree [5].

Another family of methods is based on hashing. The hash function has the objective of mapping similar objects to near positions in the hash tables. The use of multiple hashing functions increases the probability of finding the true nearest neighbor, but also increases the storage cost to save the hash tables. Locality Sensitive Hashing (LSH) [6] is one of the best known methods of this group. Some variants of LSH were proposed with good results as well [7, 8].

Recently, the creation of NN graphs has attracted a lot of attention of the information retrieval community. Results reported by Harwood and Drummond [9], and Dong et al. [10], show significant gains of graph-based approaches over hash-based and space-partitioning trees techniques, in terms of search efficiency. The idea behind this approach is illustrated in Figure 1, with a pseudo dataset of flowers, where each flower is associated with the most similar ones taking into account their color. Greedy search on nearest neighbor graphs is conduced by moving toward to the closest neighbor to the query in a sequence of steps. To determine which

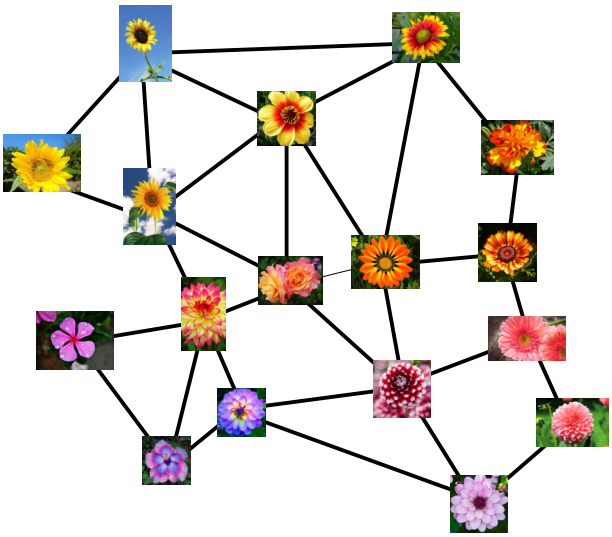


Figure 1: Example of nearest neighbor graph.

is the closest neighbor, existing methods perform an exhaustive search in the neighborhood (brute force). This can slow the process of convergence to high recall values when the graph is dense. The algorithms presented in this proposal are related to this family of approaches.

1.1 Research goals

Search over NN graphs has been shown considerable gains in efficiency at high values of precision in tasks related to finding similar objects in a given collection. However, the creation of exact NN graphs, that is, graphs where each vertex is linked to the true K nearest vertices in the whole collection, becomes prohibitively expensive in large collections, because of it implies an exhaustive exploration of the collection for every vertex. Also, it is important to keep the graph sparse to reduce the number of calculation needed to reach the true nearest points, so the selection of K is important. Another desirable property of NN graphs is that neighbors of each vertex should not be concentrated at the same region of the feature space, to be able to better cover the feature space and allow a fast navigation on the graph between any pair of vertices.

In this project, we propose to develop a novel graph-based framework called Hierarchical-Clustering-based Nearest Neighbor Graph (HCNNG), for implementing efficient NN graph construction and fast ANNS. The methodology starts by applying multiple times the hierarchical

clustering procedure over the set of feature vectors corresponding to the collection objects. Then, for each cluster found, it is created a fully connected graph over its objects. The final graph is obtained by combining all the sub-complete graphs. This initial graph could present vertices with high degree, depending on the number of execution of the clustering procedure, the clusters' size, and the distribution of the data on the feature space. Therefore, a final refinement of the graph is applied by removing some edges to keep the graph sparse. Also, we want to investigate techniques for automatic selection of parameters.

The greedy approach for searching in NN graphs, starts in a random vertex and in each step moves towards the neighbor closer to the query until some stop criterion is reached. Usually, in each step, to determine which is the closer neighbor, all neighbors are examined (compute distances to all of them). It can be used some heuristics (e.g., geometric properties) to reduce the number of neighbors examined in each step. One strategy for example, consists in examining the neighbors in the same quadrant as the query, or using some learning technique to get an indicator of which nodes are in the *direction* of the query. As part of this project, we intend to investigate and evaluate heuristics and learning techniques to reduce the number distance calculations in the search algorithm. Also, we want to investigate techniques for non randomized selection of starting vertex for search in NN graphs.

1.2 Organization of the text

The rest of the document is organized as follows. Section 2 presents related work. Section 3 describes the proposed methodology and the strategy for validation. The work plan for the remaining activities and their schedule are presented in Section 4.

2 Related Work

Related work for ANNS can be grouped into three families of approaches: tree partitioning, hash-based, and NN-graph-based methods. We introduce them in the next sections.

2.1 Tree partitioning approaches

A broadly strategy studied in the literature for nearest neighbor search is to organize the data in form of a tree. The data is partitioned into subsets, based on some criteria, at each tree's level until some stop condition is satisfied, e.g., a minimum size is reached. When a search is performed, the tree is traversed from the root to the leaves, which probably contain the closest point. A backtracking approach can be used to further explore other nodes and increase the probability of finding the most similar objects.

Tree structures for exact search (e.g., KD-Tree [1], Ball Tree [2], and Cover Tree [11]) are very efficient for low dimensional data, but their performance decreases quickly when dimensionality increases. KD-Tree is one of the most cited tree structures for exact search. In the algorithm of KD-Tree index construction, at each level and in a sequential order, a dimension is selected to split the data and a point is used to better balance the division. A well-known variant of KD-Trees for ANNS is named Randomized KD-Trees [12]. In their construction, several KD-Trees are created, where the dimension to split the data is selected randomly. Then, at query time, search is performed on all the trees, and is stopped when a fixed number of leaves is explored. This method is implemented in the widely used Fast Library for Approximate Nearest Neighbors (FLANN) [4].

Muja and Lowe [3] proposed the hierarchical k-means tree, where, at each level, the data is split using the k-means algorithm into K subsets, and then the same algorithm is applied recursively to the subsets generated. The recursion is stopped when the size of the subset is less than K. In the traverse from the root, at each level, the branch with centroid closest to the query is taken. The exploration is stopped when a fixed number of nodes is visited.

Methods for trees partitioning can be roughly divided into two groups. Techniques that divide data points with respect to hyperplanes and clustering based. From the first group, we can mention KD-Trees [1, 12], PCA-Tree [13], and RP-Tree [14]. In the second, we can found hierarchical clustering tree [4], hierarchical k-means tree [3], Ball Tree [2], Cover Tree [11], GNAT [15], and VP Tree [5].

2.2 Hash-based approaches

The principle of hash-based techniques is to use a hashing function that generates near hash values for similar objects. Thus, low-dimensional mapped values can be used to perform efficient search. One of first works was presented by Indyk and Motwani [16] and now, maybe, the most cited method is Locality Sensitive Hashing (LSH) [6]. In this technique, multiple hashing tables are created and are used at the same time to obtain a reduced list of candidates for each function. Next, an exhaustive search is performed in all the list of candidates to find the nearest neighbors. Thus, the more hash tables are created, the higher the probability to determine the nearest points. However, the number of hash tables that can be created is limited by memory.

Lv et al. [7] proposed Multi-Probe LSH, a method that reduces the high storage requirement by decreasing the number of hashing tables. Their idea is based on the supposition that if a nearest neighbor is not in the same bucket as the query, then is highly probable that it is contained in close buckets. In this way, the algorithm makes a harder exploration of closest buckets and reduces the number of hash tables needed to achieve high recall values. Bawa et al. [8] proposed a variant of LSH, which self tunes its parameters to the data.

2.3 NN-graphs-based approaches

The main idea of nearest-neighbors-graph-based approaches is to create a graph where each vertex represents an object, and edges are created between a vertex and its closest neighbors (under some similarity criterion). Then, a greedy search algorithm can be applied at query time, which starts in a random vertex and, at each step, moves to the closest neighbor to the query, until a convergence stopping criterion is reached (e.g., there is no neighbor vertex closer to the query than the actual vertex).

Harwood and Drummond [9] proposed an incremental algorithm to create an NN graph, called Fast Approximate Nearest Neighbour Graphs (FANNG). Initially, the set of vertices is composed of each object in the collection. Then, in each iteration, two vertices v_1 and v_2 are selected randomly, and a greedy search is performed using v_1 as starting vertex and v_2 as query. If the search fails to arrive at v_2, an edge is added between the last node visited and v_2. This process is repeated until the algorithm achieves a 90% success rate over a sufficiently large

number of calls to the greedy search algorithm. An important strategy employed refers to the deletion of redundant edges (edges that *occludes* others). This makes the graph to preserve the closest and spread neighbors of each vertex, leading to a more efficient graph traversal. A final improvement to the graph is obtained by adding new edges from each vertex to the set of their nearest neighbors found by using the greedy search algorithm on the graph. Next, the edges are ordered by weight (distance) and added to the final graph disregarding the redundant ones. The search over the resulting graph is performed with a variation of greedy search, applying backtracking to further explore the graph when local optimum is reached, and limiting the number of distance calculations.

Malkov et al. [17] introduced the Small World Graphs (SW). Similar to the FANNG construction algorithm, initially it is created an empty graph (no vertices or edges). In each step, a new vertex (object collection) is selected and a search over the actual graph to find a fixed number of its nearest neighbors is performed. The new vertex is added to the graph and non-directed edges are created between this vertex and the set of nearest neighbors found. This is repeated until all collection objects are included in the graph. Their objective is to create an approximation to the Delaunay graph [18], and, at the same time, maintain "long" edges to allow logarithmic navigation on the graph. This property is known as *Small World* [19]. The resulting graph presents good navigable properties, but many of the vertices end with high degree, increasing the number of distance calculations to reach the nearest neighbors.

A recent proposed approach, Hierarchical Navigable Small World (HNSW) [20], creates a hierarchy of NN graphs, in which every collection object is assigned to a maximum hierarchy level. Long edges (edges that connect distant objects) are presented in top layers, and the short ones in bottom layers. The construction of graphs is analog to SW. Incrementally, objects are added into the graphs, starting in the graph at the object's maximum level and descending up to the graph in the ground layer, linking them to the nearest ones in each level. At query time, search starts in some vertex in the top layer's graph and traverse the graph to find the closest ones to the query. When a local optimum is reached, one level is descended in the hierarchy, and search is started using as starting points the nearest vertices found at the above level. This is repeated until the ground layer is reached.

Unlike to the incremental strategy used by the initiatives mentioned above, the algorithm for graph construction proposed by Dong et al. [10] (KGraph) initially assumes a random set of neighbors for every vertex. In each iteration, based on the heuristic *a neighbor of a neighbor is also likely to be a neighbor*, these sets are updated by selecting the nearest points from the actual set and the neighbors of neighbors. The process is repeated until the sets of neighbors of each vertex do not change significantly. Although the construction algorithm converges fast and approximates with high recall the real nearest neighbors of each object collection, keeping the real nearest neighbors for each vertex could lead to store redundant edges in some cases, i.e., neighbors could be concentrated in the same region of the feature spare, slowing the convergence to high recalls values at query time.

Most of the algorithms used to search on nearest neighbors graphs combine variations of greedy search with backtracking with multiple random-starting search results to increase the probability of finding the true nearest neighbors. Results presented in described previous works using a recent benchmark[1] have shown a fast convergence of nearest-neighbors-graphs-based techniques to reach high recall values when compared with tree space partitioning techniques and hash-based methods.

Differently from above, the proposed graph construction uses the divide-and-conquer strategy to create subgraphs in reduced sets of close points (clusters), and then join all subgraphs to create a single global NN graph. At query time, rather than take a random or a fixed vertex to start the search, we choose a vertex with good probability to be near to the query point, and at early steps of search, neighbors of vertices are not fully explored by using heuristics based in a geometric property of Euclidean space.

3 Methodology

This section introduces the proposed framework for ANNS, denoted as HCNNG, which is based on a NN graph construction, resulting after performing multiple hierarchical clustering procedures (Section 3.1), and on the use of a guided search in the created graph (Section 3.2).

[1] ANN Benchmark: `https://github.com/erikbern/ann-benchmarks` (As of August 2017).

```
Function hierarchical_clustering(P, size)
    Data: data points P, min size of clusters size
    Result: directed graph edges E
    N ← |P|
    E ← φ
    if N < size then
        /* create a complete graph                                    */
        forall P_i in P do
            forall P_j in P do
                if i ≠ j then
                    add to E an edge from P_i to P_j
    else
        select randomly P_1 and P_2 points from P
        sub_P_1 ← points in P nearest to P_1
        sub_P_2 ← points in P nearest to P_2
        E_1 ← hierarchical_clustering(sub_P_1, size)
        E_2 ← hierarchical_clustering(sub_P_2, size)
        E ← E_1 ∪ E_2
    return E
```

Algorithm 1: Hierarchical clustering procedure.

3.1 Graph construction

The graph construction process relies on two steps: the execution of multiple hierarchical clustering procedures and the fusion of graphs defined by obtained clusters. The objective is to generate a graph with a good connectivity among data points. We need enough connections among the vertices for supporting effective searches, but, at the same time, without unnecessary multiple paths which may lead to inefficient processing time. These steps are detailed in the following.

The hierarchical clustering performed over a set of points defines an implicit relationship of proximity between the points in each cluster. We will explore this idea to create a proximity graph. In a hierarchical clustering procedure, when a minimum cluster size is reached, a complete subgraph is created over the actual set of points. Otherwise, two points are selected randomly and the hierarchical clustering procedure is applied recursively over the set of points nearest to both selected points. This process is outlined in Algorithm 1.

One expected property of the final graph is a high connectivity. By applying the clustering procedure just once, the resulting graph will be highly disconnected. We address this problem by

Function *create_proximity_graph(P, iterations, size)*
Data: data points P, number of random clusterings *iterations*, min size of clusters *size*
Result: directed graph edges E
$E \leftarrow \phi$
for $i \leftarrow 1$ **to** *iterations* **do**
$E_i \leftarrow hierarchical\_clustering(P, size)$
$E \leftarrow E \cup E_i$
remove all redundant edges from E **return** E

Algorithm 2: Fusion of graphs.

performing multiple random clustering procedures and combining their resulting graphs. This also solves the problem associated with points located on the border of two or more clusters, i.e., these points may belong to different clusters despite being close in the feature space.

More formally, the graph fusion step can be defined as follows. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ be a set with n clusters defined after performing multiple hierarchical clustering procedures. For each cluster $c_i \in \mathcal{C}$, we can create a proximity graph $G_i = (V_i, E_i)$, where V_i is a set of vertex defined by the points belonging to c_i, and E_i is a set of edges. Graph G_i is complete, i.e., for all vertices v_k and v_l ($v_k \in V_i$, $v_l \in V_i$, and $k \neq l$), there is an edge $e_{k,l} \in E_i$. The fusion graph G' is defined as $G' = (V', E')$, such that $V' = \cup_{i=1}^{n} V_i$ and $E' = \cup_{i=1}^{n} E_i$.

The graph resulting from the fusion of multiple clustering results may become too dense (vertices with high degree), which will increase the time of navigation due to the need for computing the distances from all neighbors to the query at each step of the search. Most of the edges in the merged graph keep redundant information, in the sense that we can reach their final vertices through multiple paths. Multiple techniques can be applied to address this problem. For example, in the euclidean space, it could be used the concept of *occlusion* of edges proposed by Harwood and Drummond [9] to remove the redundant edges in the combined graph. The process to create the final proximity graph is outlined in Algorithm 2.

A visual example of all the hierarchical clustering process described above is illustrated in Figure 2. Sub-figures (a), (b), and (c) show the graph obtained after performing three independent hierarchical clustering procedures. Sub-figure (d) shows the graph resulting from the combination of these three graphs, and, as expected, its vertices have a high degree. Sub-figure (e) shows the graph resulting after removing redundant edges. We can observe that by removing redundant edges, we still keep a good connectivity among the vertices and possibly a

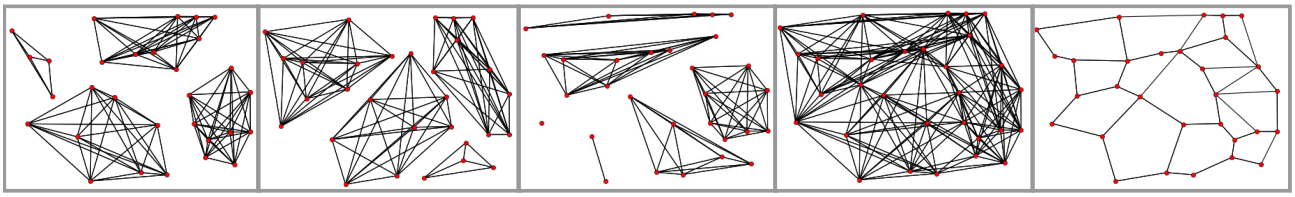


Figure 2: Proximity graph created over thirty two-dimensional random points, using 3 clusterings and a cluster minimum size equal to 10.

fast-to-traverse sparse graph.

We choose this hierarchical-clustering-based approach instead of other methods like k-means, because it is hard to estimate the initial number of clusters and the number of iterations, without previous knowledge of the data. It is easier to set the size of each cluster based on the number of neighbors we want to connect each vertex. Also, the time complexity of each clustering process scales in a logarithmic factor to the collection size leading to a total of $O(Ndlog(N/n))$, plus $O(Ndn)$ to construct each graph, where N is the size of the collection, d the data dimensionality, and n the estimated size of the clusters.

3.2 Guided search

The greedy approach to perform a search over a graph of nearest neighbors consist in, randomly, selecting a starting vertex and then, in each step, moving to the neighbor closer to the query. This is repeated until there is no neighbor closer to the query than the actual vertex. Some variants were proposed, as by Harwood and Drummond [9], using a priority queue to save the visited points and apply backtracking when a minimum local is reached. Also, Malkov et al. [17] proposed the use of multiple random starting points. In both cases, neighbors of each vertex visited are fully explored, that is, distances are computed from all neighbors to the query.

The main idea of the proposed *guided search* is to avoid computing exhaustively the distance from the query to all neighbors in each step of the search. Our objective is to focus the graph traversal process (search) on a reduced set of neighbors that are probably in the *direction* to the query. An initial heuristic we intent to evaluate is based on the concept of quadrants . Figure 3 illustrates this idea. At any point in the navigation (say u), we will just compute the distances from the query (Q) to those neighbors located at the same quadrant (v) as the query

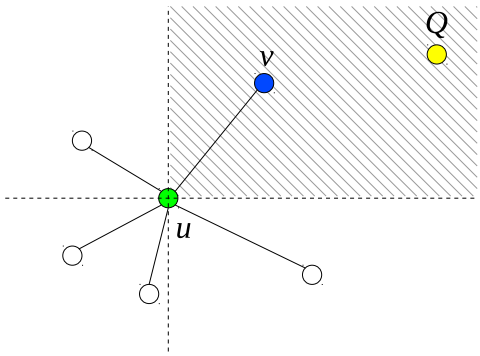


Figure 3: Example of directed selection of neighbors.

(quadrant highlighted in the figure), in reference to the actual vertex.

Figure 4 shows an example of multiple steps of a guided search from the starting point (green vertex) to the query point (yellow star), where the vertices explored by guided search are colored in blue and green (total 5), and vertices explored by classical greedy approach are those colored in gray in addition to the blue and green ones (total 10). The guided search starts at vertex 11 and, in the first step, it is computed the distance from the query to just vertices 6 and 8. Vertex 8 is then selected for the next step. In the second step, it is explored vertex 5, which is selected for the next step, as it is closer to the query than vertex 8. Finally, vertex 3 is explored, and as it is not closer to the query than the actual vertex, the search ends. As we can observe, the number of distance computations is decreased in half compared to classical greedy approach, in this example.

Determining the neighbors located in the same quadrant, at query time, is almost as expensive as computing all the distances, so it will be necessary a preprocessing stage (offline), to be applied on each vertex neighborhood, with the objective of organizing the neighbors of each vertex in quadrants (subspaces), taking as origin the vertex itself. Then, at query time, we can use a slightly modified version of backtracking search described by Harwood and Drummond [9]. We start at a random vertex and, for each step, instead of exploring all neighbors, we continue the search through the neighbors in the same subspace as the query. When is not found a closer point to the query in the selected subspace, we have to explore other adjacent subspaces.

On the other hand, we can also use learning techniques to choose the neighbors to be

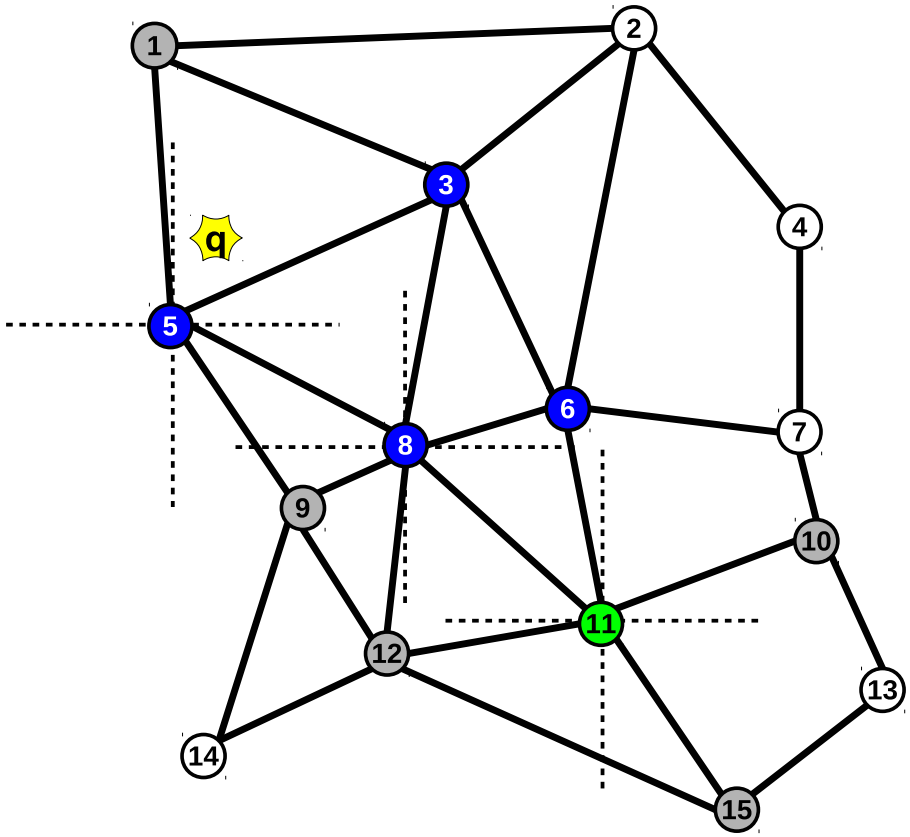


Figure 4: Example of multiple steps in the guided search.

examined in each step of the search. For example, it could be used Genetic Programming [21] to discover a map function to convert each feature vector to 1-Dimensional vectors (a number) and use them to examine the neighbors with respect to the distance to query's mapped value. We intend to investigate and evaluate this and other novel ideas based on machine learning techniques as part of this project.

3.3 Validation

This section presents an initial description of experimental protocol we plan to use to validate the proposed framework.

3.3.1 Datasets

We intend to experiment, among others, with two bases from BIGANN datasets[2] for ANNS. One collection is composed of 1 million SIFT features (128 dimensions) to index construction, and 10K queries to evaluate performance. The other collection contains 1 million GIST features (960 dimensions) and 1 thousand queries. Also, these datasets were previously used in other works [9, 22, 23] to evaluate ANNS techniques.

3.3.2 Baselines

We will compare the proposed approach, with several well-known and recent methods of the state of the art. FLANN library [4] is probably the most known library for ANNS. There are three principal space partitioning trees techniques implemented in FLANN: Randomized KD-Trees [12], K-Means Tree [3], and Hierarchical Clustering Tree [24]. This library also contains an auto-tunned algorithm, which selects the best algorithm (included in FLANN) and parameter values based on the data.

Among the nearest-neighbors-graph-based techniques, we will include as baseline a recently proposed method [9], called FANNG. We will use our own implementation, and run the experiments with parameters reported by authors on the same datasets. Another baseline included from this family of methods was SW [17]. The experiments will be conduced with the implementation found in *ANN-Benchmarks*. Also, we want to compare with KGraph [10] technique, using the implementation provided by authors in their website.[3]

3.3.3 Evaluation criteria

We plan to employ widely-used evaluation measures for ANNS like the Speedup $\times$ Recall charts. To keep the speedup independent from architecture where experiments are executed, we will only consider the number of distance calculations performed by each method. Thus, speedup is defined by:

[2]BIGANN: `http://corpus-texmex.irisa.fr/` (As of August 2017).
[3]KGraph: `http://www.kgraph.org` (As of August 2017).

$$Speedup = \frac{Collection\ size}{Number\ of\ distance\ calculations}$$

The recall is defined by the fraction of true nearest neighbors that are successfully retrieved. Two classical experiments are performed to evaluate ANNS methods, one to search the nearest neighbor (1-NN) and the other to search the 10 nearest neighbors (10-NN).

We can also evaluate the graph structure by means of unsupervised measures, as the *navigability*. The navigability of a NN graph can be defined as the probability of successfully reaching, from any vertex, any other using a greedy search algorithm. Although it could be a indicator of a good NN graph structure, if we only consider this value, then a complete graph would have a perfect score, but actually this is not an appropriate graph, given that the NN search becomes a linear search on a complete graph. Thus, we intent to investigate novel unsupervised evaluation measures to evaluate the graph structure considering both the navigability and the expected number of distance calculations in a NN graph.

4 Work plan and schedule

The work plan consists of activities in the following topics, in accordance to the schedule presented in Table 1.

1. Courses and initial literature review.
2. NN graph construction algorithm.
3. Techniques for removing edge redundancy and reinforce connectivity on NN graphs.
4. Heuristics for guided search on NN Graphs.
5. Learning techniques for guided search on NN Graphs.
6. Internship (PhD sandwich program with BEPE scholarship).
7. Validation of framework and publication of the main results.
8. Writing and defense of the PhD work.

Note that this work has started in March 2016. The first year was dedicated to the courses and initial literature review. We are currently investigating techniques for improving connectivity on initial NN graph and testing some geometrical heuristics for guided search on NN graphs.

Table 1: Schedule of planned activities.

Activity	Semester							
	1s2016	2s2016	1s2017	2s2017	1s2018	2s2018	1s2019	2s2019
1	•	•						
2		•	•					
3			•	•	•			
4			•	•	•			
5					•	•	•	
6					•	•		
7					•	•	•	•
8								•

References

[1] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[2] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[3] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'2009)*. INSTICC Press, 2009, pp. 331–340.

[4] ——, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[5] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA'1993. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1993, pp. 311–321.

[6] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communication of the ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008.

[7] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB'2007. VLDB Endowment, 2007, pp. 950–961.

[8] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: Self-tuning indexes for similarity search," in *Proceedings of the 14th International Conference on World Wide Web*, ser. WWW'2005. New York, NY, USA: ACM, 2005, pp. 651–660.

[9] B. Harwood and T. Drummond, "FANNG: Fast approximate nearest neighbour graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR'2016, 2016, pp. 5713–5722.

[10] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW'2011. New York, NY, USA: ACM, 2011, pp. 577–586.

[11] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML'2006. New York, NY, USA: ACM, 2006, pp. 97–104.

[12] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR'2008, 2008, pp. 1–8.

[13] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, no. 1, pp. 579–589, 1991.

[14] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, ser. STOC'2008. New York, NY, USA: ACM, 2008, pp. 537–546.

[15] S. Brin, "Near neighbor search in large metric spaces," in *Proceedings of the 21th International Conference on Very Large Data Bases*, ser. VLDB'1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 574–584.

[16] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, ser. STOC'1998. New York, NY, USA: ACM, 1998, pp. 604–613.

[17] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[18] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[19] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, ser. STOC'2000. New York, NY, USA: ACM, 2000, pp. 163–170.

[20] Y. A. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *arXiv preprint arXiv:1603.09320*, 2016.

[21] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[22] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[23] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[24] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Proceedings of the 2012 9th Conference on Computer and Robot Vision*, ser. CRV'2012. Washington, DC, USA: IEEE Computer Society, 2012, pp. 404–410.