Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

A Genetic Programming Approach for Searching on Nearest Neighbors Graphs

Javier A. Vargas Muñoz · Zanoni Dias · Ricardo da Silva Torres

Received: date / Accepted: date

Abstract Nearest neighbors graphs have gained a lot of attention from the information retrieval community since they were demonstrated to outperform classical approaches in the task of approximate nearest neighbor search. These approaches, firstly, index feature vectors by using a graph-based data structure. Then, for a given query, the search is performed by traversing the graph in a greedy-way, moving in each step towards the neighbor of the current vertex that is closer to the query (based on a distance function). However, local topological properties of vertices could be also considered at the moment of deciding the next vertex to be explored. In this work, we introduce a Genetic Programming framework that combines topological properties along with the distance to the query, aiming to improve the selection of the next vertex in each step of graph traversal and, therefore, reduce the number of vertices explored (scan rate) to find the true nearest neighbors. Experimental results, conducted over three large collections of feature vectors and four different graph-based techniques, show significant gains of the proposed approach over classic graph-based search algorithms.

Keywords Distance learning · Genetic programming · Image retrieval · Graph-based indexes · Nearest neighbor searches · Network theory

Javier A. Vargas Muñoz*
University of Campinas
E-mail: javier.munoz@ic.unicamp.br

Zanoni Dias
University of Campinas
E-mail: zdias@g.unicamp.br

Ricardo da Silva Torres
Norwegian University of Science and Technology
E-mail: ricardo.torres@ntnu.no

1 Introduction

A high amount of multimedia content is being generated every second due to the large use of low-cost portable devices and the growth of internet access in the last decades. This leads to the creation of large collections of multimedia content, and, at the same time, to the need for mechanisms that, efficiently, support the task of retrieving relevant information from these collections. Advances in multimedia representation have allowed the design of techniques that map multimedia objects (e.g., image, text, audio, and video) into feature vector spaces (commonly in the scale of hundreds or thousand of dimensions) fostering the creation of effective search systems.

In this context, the Nearest Neighbors (NN) search is a problem broadly studied in the literature. This problem consists in searching for the nearest points to a query from a set of points with arbitrary dimensionality (that represents multimedia objects), considering a given distance function. There are two well-known variants of this problem: the K-Nearest Neighbors and Range Queries. The first consists in searching for the K points that are closer to the query than any other point in the set. The second aims to find all points that are inside of the hypersphere with the query as center and radius r. This paper refers only to the first problem (often referred to as K-NN search).

A straightforward solution to this problem is the linear search algorithm, which scans the entire collection of feature vectors (objects) and returns those with the lowest distance to the query point. However, this solution is not scalable in scenarios with millions of vectors and high concurrency of queries. Alternative data structures were proposed for efficient exact searches yielding a logarithmic time complexity (e.g., KD-Tree [9] and R-Tree [20]), although this logarithmic property is only guarantied for low dimensional data. More accurate methods for representing multimedia objects usually employ high dimensional data. For this search scenario, many approximate approaches were proposed to assess the efficiency of NN searches at the cost of losing precision of results. These last approximate techniques can be roughly divided into four groups: space-partitioning trees [13, 37, 44, 45, 54], hashing-based [3, 8, 23, 27, 34, 53], quantization [4, 7, 11, 16, 26, 28, 31, 52], and nearest neighbor graphs [14, 21, 24, 35, 36].

In recent works [5] and benchmarks,[1] NN-graph-based approaches have shown to outperform consistently classical and recent approximate approaches for NN search. In NN-graph-based techniques, firstly, all feature vectors of the collection are indexed by means of a graph, where each feature vector is associated with a unique vertex and connected with the other near ones. Then, the search of NN is performed by traversing the graph in a greedy way, with similar strategy to the *hill climbing* optimization procedure [39], starting in some vertex, and, in each step, selecting the neighbor of the current vertex that is closer to the query (based on some distance function) as the next vertex to be explored. Methods for the construction of these graphs differ considerably, but

[1] ANN benchmark: `https://github.com/erikbern/ann-benchmarks` (As of June 2021).

the search algorithm employed in these works did not change significantly from the previous idea, and all of them select the next vertex in the graph traversal based only on the distance to the query. The selection of the next vertex to be explored plays an important role at converging efficiently to the true nearest neighbors. Other kinds of properties of vertices could help improving the selection of the next vertex, such as, the local topological properties of vertices. For example, if one vertex has a very high degree, then we could avoid exploring it, since it would be very costly to visit all its neighbors.

In the context of networks, there are many metrics [12] broadly studied in the literature that capture different properties of the topology of vertices' neighborhood. In this work, we propose a Genetic Programming (GP) framework to find a near-optimal scoring function that takes into consideration both the classical distance-to-the-query and local topological properties of vertices. The scores obtained by this function are used to sort the vertices aiming to minimize the number of vertices explored to discover the true nearest neighbors. GP has been shown to perform well in optimization scenarios like this [2, 10, 30, 43]. In the context of the GP approach, an individual from the population corresponds to a candidate scoring function.

1.1 Contributions

In this work we explored a novel idea to exploit different topological properties of vertices in order to improve the navigation on NN graphs at search time. The main contributions of these work are listed bellow:

- We introduce a novel Genetic Programming (GP) framework that aims to discover a near-optimal combination of local topological properties of vertices along with the classical *distance-to-the-query*, which improves the criterion for selection of the next vertex to be explored in the search algorithm, leading to a more efficient discovery of the true NN.
- The proposed formulation is domain-agnostic. In the training phase, the GP framework is capable of determining suitable combination functions that capture the overall properties of a given dataset. Those functions are expected to lead to more efficient searches.
- We validated our search approach on a recent technique for NN graph construction [51], along with the three other previous techniques considered [50], showing the consistency of our results independently of the dataset and the NN graph construction technique. Also, parts of this work were presented in the PhD thesis of the first author [49].

This paper is an extension of our previous work [50]. Different from that paper, we included two new topological properties to the GP framework that improved our previous results. Furthermore, we experimented on two other collections of image feature vectors, in which our search approach demonstrated to outperform classical search algorithms as in the case of the collection of textual feature vectors.

In the next section, background concepts about NN graphs and Genetic Programming are presented. Those concepts will be employed in Section 3 to describe our approach. In Section 4, the experimental protocol adopted to validate the performance of our approach and the experimental results are detailed. Finally, Section 5 concludes the paper and presents future research directions.

2 Background Concepts

This section provides an overview of NN graphs and Genetic Programming.

2.1 Nearest Neighbor Graphs

NN graphs present an easy-to-understand scheme for multimedia indexing. Each multimedia collection object is represented as a vertex in the graph, and each of them is connected to the most similar ones, i.e., to those objects whose feature vectors are the closest ones. The naïve algorithm for the construction of NN graphs consists in, for each vertex, scanning the whole collection and selecting the K-nearest vertices as their neighbors. Therefore, this search scheme results in a quadratic solution at the collection size. In real scenarios with millions of feature vectors, this naïve algorithm is unpractical. In the following, we describe some efficient approaches that have been proposed to speed up the creation of approximate NN graphs. Also, we provide details on how searches are performed over those graphs.

2.1.1 Creation of NN graphs

NN graphs have also been used in other tasks, such as image annotation [22, 32, 46, 47]. In this section, however, we only considered those graph construction techniques that were employed in the task of NN search. There are two approaches commonly employed for constructing NN graphs, which are illustrated in Fig. 1. In the first approach, sub-figure (a), an initial graph is constructed (could be randomly) to connect vertices, and, then, at each iteration, the neighbors of each vertex are updated, in order to discover closer or more adequate neighbors. In the second approach, sub-figure (b), vertices and/or edges are added incrementally to the current graph, that could be initialized as an empty graph with neither edges nor vertices. Many approaches have been proposed to construct approximate NN graphs based on these ideas. In the next paragraphs, we briefly describe some state-of-the-art techniques that we employed in this work as baselines methods.

KGraph [14]: This method creates an initial graph containing all vertices, where each vertex is connected to a list of K randomly selected vertices. Then, iteratively, based on the simple assumption that "*a neighbor of a neighbor is also likely to be a neighbor*," the method improves the list of neighbors of each

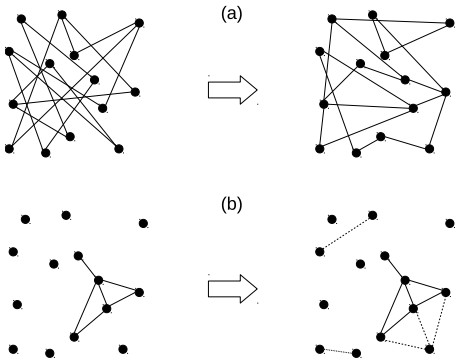

Fig. 1 Illustration of the two typical approaches for NN graph construction. (a) initial graph optimization; (b) incremental construction by adding edges and/or vertices.

vertex by exploring the neighbors of their neighbors and replacing them by those which are closer to some vertex in the current list. The objective is to discover the set of true K-nearest neighbors for each vertex. This optimization process stops when the list of neighbors of all vertices does not change significantly.

SW-graph [35]: Initially, the graph has neither vertices nor edges. Every vertex is inserted into the graph, one at a time. When one vertex is inserted, it is performed a K-NN search on the current graph, with an algorithm similar to that one described in the next section. Then, undirected edges, between that vertex and every K-NN found, are created. The graph construction ends when all vertices are inserted.

HNSW [36]: This algorithm for graph construction is similar to the one used for the SW-graph, since it also uses the search algorithm to find the vertices (in the current graph) that will be connected to the new inserted vertex. However, this algorithm creates a hierarchy of graphs, where edges in top layers correspond to long edges (edges that connect distant vertices), while bottom layers contain short edges (edges that connect closer vertices). A new vertex is inserted first in the graph of a random layer, then in the graph of the layer below this, and so on, until the ground layer is reached.

FANNG [21]: Differently from the techniques above, initially the graph contains all vertices but no edges. A simple process is repeated though a number of iterations. At each iteration, a greedy search is performed from one vertex (start) towards another (target), both selected randomly. If the target vertex is not reached successfully, then a directed edge is created from the last vertex visited to the target vertex. Next, this process is applied in the opposite direction, from the target to the last vertex visited. Otherwise, nothing is altered on the graph. This process is repeated until a maximum number of searches are performed or a rate of successful searches is reached.

HCNNG [51]: To create the NN graph, this method performs multiple hierarchical clustering executions, then, on each cluster created, the points inside them are connected through a Minimum Spanning Tree (MST). Finally,

```
1  Function SearchNN(G, q, T)
       Data: NN graph G, query point q
       Data: maximum distance calculations T
       Result: nearest vertex n, nearest vertex distance d
2      n ← some vertex in G
3      d ← distance(n, q)
4      Q ← initialize priority queue with tuple [n, d]
5      while T > 0 do
6          v ← Q.pop()
7          foreach u ∈ Neighbors(v) do
8              if T > 0 and u not visited then
9                  d* ← distance(u, q)
10                 Q.push([u, d*])
11                 T ← T − 1
12                 if d* < d then
13                     n ← u
14                     d ← d*
15     return n, d
```

Algorithm 1: Algorithm for search in NN graphs.

the method merges all the MSTs and generates a global NN graph for all the points.

2.1.2 Search in NN graphs

As seen in previous section, techniques for NN graph construction are based on conceptually different strategies. However, most of search algorithms used on these NN graphs rely on roughly the same strategy. This strategy is detailed in Algorithm 1. The traversing starts by initializing the global minimum (nearest neighbor) at some vertex on the graph (lines 2-3), and a priority queue to help to select the next vertex to be explored (line 4) at each step (an iteration of loop in line 5), this is, the vertex in the queue with the minimum distance to the query. In each step of traversal, after the next vertex is selected (line 6), the neighbors of this vertex are scanned (line 7). For any neighbor, if it was not visited previously (line 8), then it is pushed to the queue (lines 9-10). Also, if any neighbor is closer to the query than the global minimum, then it is updated (lines 12-14). When the maximum number of vertices to be explored is reached, the global minimum discovered is returned (line 15).

An example is illustrated in Fig. 2, with starting vertex v_0 and query point q (red point). The first vertex is taken from the queue (v_0), then, their neighbors are explored and pushed to the queue. In the next step, from the vertices on the queue, the closest to the query is taken and explored (v_1), and so on until the maximum number of distance calculation is reached. Note that the definition of the next vertex to be explored does not depend only on the neighbors of the current vertex, as all vertices on the queue are candidates (blue vertices, after the 3rd step).

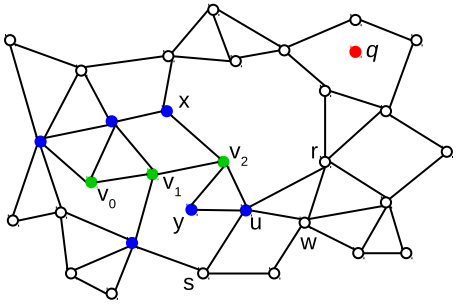

Fig. 2 Example of search using Algorithm 1.

Malkov *et al.* [35] presented a modified version of this algorithm, which considers a different stop condition. According to their proposal, a graph is traversed until a set of K-nearest neighbors remains unchanged at a given iteration. Also, they proposed to perform multiple searches, with different starting vertices, and then combine the results of such searches to return the best k vertices. Harwood and Drummond [21] also proposed to use the nearest vertex to the data mass center as starting vertex for search. Differently from previous ideas, Vargas *et al.* [51] introduced a pruning strategy to avoid scanning all neighbors at any step of traversing, just those neighbors in the "direction" of the query.

The traversal strategies employed on graphs for NN searches, as those mentioned above, are not directly related to the well-known strategies for search on graphs/networks, like the A^* [17] or other *landmark*-based algorithms [15,19,42,48], since those techniques were devised for the approximate shortest path problem between two vertices on a graph (P2P). The definition of *distance* is different in both problems. In the context of NN search, the distance between two vertices is defined (commonly) as the Euclidean distance between their respective feature vectors. On the other hand, for P2P, the distance is defined as the shortest path between two vertices (sum of the edge weights contained in the path). Also, in the NN search problem, it does not matter the cost of the path taken, since it leads to an early discovering of the vertices with the minimum distance (between their feature vectors) to the query, which in most of cases (the query) is represented by an unseen vector on the graph construction phase. Therefore, solutions for those problems cannot be mapped to each other.

In this paper, we adopt the conventional version of the search algorithm in our experiments, shown in Algorithm 1, as it allows us to control the scan rate of a graph through the parameter T. Note that this algorithm can be easily adapted to search for the K-nearest neighbors, by simply adding a list to maintain the K-nearest neighbors seen in the whole traversing process.

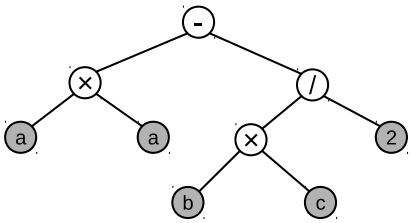

Fig. 3 Example of tree representation of a GP individual.

2.2 Genetic Programming

Genetic Programming (GP), introduced first by Koza [29], is a problem solving technique inspired in the biological inheritance and evolution. Each potential solution is called *individual*, and is commonly represented by a tree. Fig. 3 presents a tree representation of the mathematical function $f(a, b, c) = a \times a - ((b \times c)/2)$. The whole GP process to discovery near-optimal solutions is detailed in the following steps:

1. **Create initial population**: initially, a population of fixed size is created, where each tree (individual) is defined randomly. A tree is composed of functions (white nodes in Fig. 3) and terminals (gray nodes). Functions are the internal nodes, employed to combine the terminals, usually mathematical functions. Terminals are the leaf nodes, employed as inputs of individuals.
2. **Evolve population**: through a number of generations, individuals are evolved by employing genetic operators. The following steps are repeated at each generation.
   2.1. **Compute fitness of individuals**: at the beginning of each generation, the fitness value is computed for all individuals. The fitness value is an indicator of how well an individual performs in a given task. Therefore, this function is application dependent.
   2.2. **Select individuals for genetics operators**: an important factor towards the convergence of the evolution process is the technique employed for selecting individuals that will be used for genetic operators. There are many techniques proposed in the literature [25]. Usually, these approaches are based on the fitness value of individuals or on their relative order in the whole population (rank-based).
   2.3. **Apply genetic operators**: genetic operators are applied on selected individuals to create individuals for the next generation. The following operators are commonly used:
      i. **Reproduction**: this operator takes a percentage of the individuals with best fitness values and copies them directly to the next generation. This operator guaranties that the best individual of a generation will be at least as good as best individual of the previous generation.

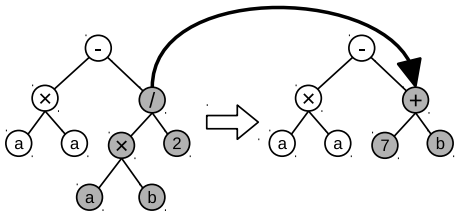

Fig. 4 Example of the mutation operator.

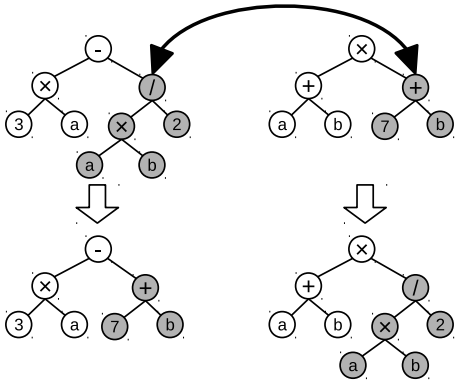

Fig. 5 Example of the crossover operator.

ii. **Mutation**: this operator is applied on a individual by taking a randomly selected subtree and replacing it with a randomly generated tree. Fig. 4 illustrates this operator. The purpose of mutation is to add a minimum of diversity to the population.
iii. **Crossover**: this operator takes two individuals and exchanges two randomly selected subtrees. An example is shown in Fig. 5. The objective of crossover is to create new diverse individuals by exchanging genetic information from parents.

2.4. **Replace old population**: the population of the next generation is composed of individuals created in step 2.3, then, the evolutionary process returns to step 2.1.

3. **Select best individuals**: the fitness of individuals of the last generation is computed, and, then, the best individuals are returned.

3 GP-based Graph-based Searches

As shown in Algorithm 1, in the classical approach for searching on NN graphs, vertices of graph are scored in the priority queue based only on their distances to the query, so the vertex with minimum score (minimum distance) is taken from the queue in each step and its neighbors visited. However, local topological information of vertices can be used to improve the criterion for selection

of the next vertex to be explored. We explore the idea of better scoring those vertices that are expected to allow a faster discovery of the true nearest neighbors. We modeled this scoring function as a combination of the distance and topological properties through basic mathematical operators, therefore, we adopted the classical tree representation of GP individuals for mathematical functions, as detailed in the previous section.

Next sections detail the proposed approach for discovering appropriate vertex scoring functions to guide graph-based searches, highlighting its main components.

3.1 Pipeline for GP-based distance learning

An overview of the proposed framework to discovery a more suitable scoring function for search on NN graphs is shown in Fig. 6. The whole process is similar to the GP process detailed in Section 2.2.

Algorithm 2 summarizes how the whole learning process is performed. At first, an initial population of candidate solutions is created randomly (Line 2), by employing a set of functions composed of mathematical operators, and a set of terminals associated with several Search Dependent (SD) and Search Independent (SI) properties – described in the next section. Line 3 refers to the computation of the baseline search results for all queries of the training set. The goal is to maximize the difference between the effectiveness performance (recall) when the GP framework is used compared to using the L2 metric. Then, this initial population is evolved over several generations (Lines 4-10). In the beginning of each generation, the fitness of each individual is evaluated (Lines 5-7). In our formulation, we compute the fitness function based on the search performance of the scoring functions encoded as GP individuals. Therefore, given an individual $\hat{f}(x)$, to compute its fitness value, we average the search performance on a training set of queries obtained by using $\hat{f}(x)$ function in lines 3 and 9 of Algorithm 1 (i.e., the *distance* function is implemented using $\hat{f}(x)$). The exact computation of this fitness value is detailed in Section 3.3.

After the fitness evaluation, some individuals are selected (Line 8) to be subjected to genetic operators (Line 9), and, finally, a new population is created, composed of the individuals produced by the genetic operators (Line 10). At the end of this evolutionary process, the most fitted individual (scoring function $\hat{f}(x)$ in the algorithm) seen through the whole process is selected, and employed in the search algorithm to support the execution of new queries (Line 11).

Fig. 7 illustrates an example of a hypothetical individual that could be discovered by the proposed GP-based framework. This tree represents the vertex scoring function $\hat{f}(x) = \frac{(D \times D)}{P} - \frac{((N \times J)+E)}{2}$, which combines the following properties of a given vertex x: distance to the query (D), preferential attachment (P), vertex degree (N), Jaccard coefficient (J), and the edge weight (E). All these properties are described in the next section.

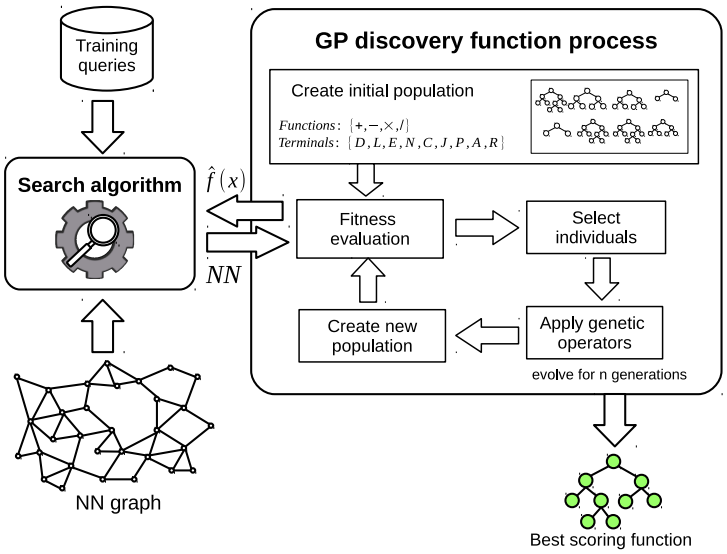

Fig. 6 GP-based framework to discover better scoring functions for NN searches.

Function *LearnScoringFunction(G, Q)*
  Data: NN graph G, training queries Q
  Result: scoring function $\hat{f}$
  $P \leftarrow$ create an initial random population of scoring functions f
  $R_{L2} \leftarrow \forall q \in Q$, search the NN on G using $L2$
    `// computation of the baseline search results`
  for *g generations* **do**
    for $f \in P$ **do**
      $R_f \leftarrow \forall q \in Q$, search the NN on G using f
      fitness$(f) \leftarrow Recall(R_f) - Recall(R_{L2})$
    $S \leftarrow$ **select** individuals from P based on fitness
    $P' \leftarrow$ **apply** genetic operators on S
    $P \leftarrow P'$
  return $\hat{f} \in P$ with max fitness value

Algorithm 2: Scoring function learning process.

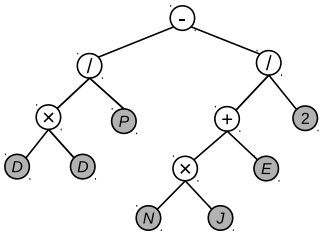

Fig. 7 Example of individual in the proposed framework.

3.2 Vertex properties

The set of properties described in the following will be used as terminals on GP individuals of the evolutionary process. They are listed in Table 1. Let v be the

Table 1 List of vertex properties employed as terminals on GP individuals.

Symbol	Property
D	Distance
L	Path length
E	Edge weight
N	Vertex degree
C	Common neighbors
J	Jaccard coefficient
P	Preferential attachment
A	Adamic Adar
R	Edge redundancy

current vertex at search that was popped from queue in line 6 of Algorithm 1, and u the neighbor of v being explored (line 7). We considered two types of properties to be taken into account by the scoring function discovery process: Search Dependent (SD) and Search Independent (SI).

Properties from the first group can only be computed at search time. We included in our method the following search dependent properties:

- **Distance (D)**: this is the traditional distance from vertex u to the query point (computed in line 9 of Algorithm 1).
- **Path length (L)**: we can adapt Algorithm 1 to keep track of the length of the path (number of vertices) from the starting vertex to u. In the example shown in Fig. 2, $L(u) = 3$ (path from v_0 to u).

On the other hand, search independent properties depend on the local topology of NN graph's vertices, thus, these properties can be pre-computed offline, adding no extra cost at search time. Let τ be the function that returns the set of neighbors of a given vertex. Thus, in Fig. 2, $\tau(v_2) = \{v_1, u, x, y\}$ and $\tau(u) = \{v_2, r, s, w, y\}$. We will use these two vertices to illustrate this type of properties. We considered the following properties from this category:

- **Edge weight (E)**: the weight of the edge between v and u. In our work, $E(v_2, u) = dist(v_2, u)$.
- **Vertex degree (N)**: the degree of u, given by $N(u) = |\tau(u)|$. For example, $N(u) = 5$.
- **Common neighbors (C)**: the number of common neighbors between vertices v and u, given by $C(v, u) = |\tau(v) \cap \tau(u)|$. For example, $C(v_2, u) = |\{y\}| = 1$.
- **Jaccard coefficient (J)**: a broadly used similarity measure commonly employed in information retrieval tasks, defined as:

$$Jaccard(v, u) = \frac{|\tau(v) \cap \tau(u)|}{|\tau(v) \cup \tau(u)|}$$

for example, $J(v_2, u) = 1/8 = 0.125$.

- **Preferential attachment (P)**: another well-known metric in networks. This metric serves as an indicator that vertices with many neighbors will create more connections in the future (in dynamic graphs). This metric is given by:

$$PrefAttach(v, u) = |\tau(v)| \times |\tau(u)|$$

  for example, $P(v_2, u) = 4 \times 5 = 20$.
- **Adamic Adar (A)**: a classical metric employed initially in the context of link prediction that is defined by:

$$AdamicAdar(v, u) = \sum_{x \in \tau(v) \cap \tau(u)} \frac{1}{\log |\tau(x)|}$$

  for example, $A(v_2, u) = \frac{1}{\log |\tau(y)|} = \frac{1}{\log(2)} = 3.32$.
- **Edge redundancy (R)**: we introduce this binary property that takes the value of 0 if there is a vertex w (different from v and u) that is neighbor of v and u, otherwise, this property takes the value of 1. For example, $R(v_2, u) = 0$, since exists a neighbor in common between v_2 and u (y).

To illustrate how the properties described above are combined, suppose that the best scoring function found through the evolution process was $\hat{f}(x) = \frac{(D \times D)}{P} - \frac{((N \times J)+E)}{2}$, and consider that v_2, u, and q in Fig. 2 represent the points $(4, 4)$, $(5, 3)$, and $(10, 10)$, respectively. Then, the score that will be associated with vertex u will be: $\hat{f}(u) = \frac{(\sqrt{72} \times \sqrt{72})}{20} - \frac{((5 \times 0.125)+\sqrt{2})}{2} = 2.58$.

We selected this set of properties based on their associated low computational cost, an important requirement, as our method is expected to support searches on large graphs.

3.3 Fitness computation

It is of paramount importance to define an adequate fitness function based on what we are aiming to optimize. As we intend to discover scoring functions that improve search performance on NN graphs, we decided to use a conventional metric on information retrieval known as *recall* to measure the search performance for a given individual $\hat{f}$. Recall@k is defined as the rate of relevant objects ranked at the first k positions.

Also, in order to have an indicator of the improvement in terms of the *recall* obtained by using $\hat{f}$ instead of the classical approach, we defined the fitness function as the difference of the recall obtained between the search based on $\hat{f}$ and the search based only on the distance to the query. Formally, this fitness function is given by:

$$fitnessNN(\hat{f}) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \Big(g(T, \hat{f}) - g(T, L2) \Big) \tag{1}$$

where the function $g(T, s)$ computes the average *recall@1* obtained on a training set of queries Q, by using s as scoring function at search, and limiting

the number of vertices explored to T. Also, $L2$ represents the function that computes the Euclidean distance between any vertex and a query. The reason why to average the gain obtained for different values of T is to discover scoring functions that perform well regardless the maximum number of vertices allowed to explore.

4 Experiments

This section presents the experimental protocol employed to validate the proposed technique, and our experimental results.

4.1 Datasets

We experimented with three different datasets, one textual and two images collections. These datasets have been extensively used in several evaluation protocols adopted in studies similar to ours [21,36,51]. The employed datasets are described next.

- **GloVe**:[2] this dataset contains vector representations of words extracted by a well-known method for word representation [40] from a corpus of 2 billions of tweets, resulting in 1.2 millions of 100-dimensional vectors. From this set, we selected randomly 1 million of vectors for graph construction, and 10 thousand as queries for search evaluation.
- **YFCC100M**:[3] the Yahoo-Flickr Creative Commons 100 Million (YFCC-100M) contains 99.2 million photos and 0.8 million videos from Flickr (we only considered the images). We employed the feature vectors provided by Popescu *et al.* [41], representing improved VLAD vectors, in which their initial dimensions (32,768) were reduced with PCA+whitening, maintaining the 128 most significant dimensions for our experiments. We selected randomly a subset of 1 million of images for graph construction and 10 thousand queries.
- **SIFT1M**:[4] a broadly used dataset introduced by Jegou et al. [26], consists in a collection of 1 million SIFT vectors for index construction and 10 thousand queries. The SIFT [33] method (from Scale-invariant Feature Transform) is a well-known algorithm for description of local features in images. SIFT vectors have 128 dimensions.

4.2 NN graph baselines

We considered four of the methods described in Section 2.1.1 for construction of NN graphs: KGraph [14], SW-graph [35], and FANNG [21], and HC-

[2] GloVe: `https://nlp.stanford.edu/projects/glove/` (As of June 2021).

[3] YFCC100M: `http://multimedia-commons.s3-website-us-west-2.amazonaws.com` (As of June 2021).

[4] BIGANN benchmark: `http://corpus-texmex.irisa.fr` (As of June 2021).

NNG [51]. In the case of KGraph, we performed experiments using the implementation provided at the authors' website.[5] For the SW-graph method, we used the implementation included in the Non-Metric Space Library (NMSLIB).[6] For FANNG, we performed experiments with our own implementation. For HCNNG, we also implemented this technique. Finally, we also included in our comparisons the HNSW method, however, we were not able to apply our proposed search approach over it, since the search employed by this technique differs significantly from the approach considered by all four methods mentioned above, due to their hierarchical structure of multiple graphs. We leave the investigation of the use of our method combined with HNSW for future work.

We decided to set the maximum vertices degree to the same value (to have a fair comparison) for the graphs created by KGraph, FANNG, and HCNNG. Selecting a low value for maximum degree could lead to a disconnected graph, so we tested with many cut-off values near to the average degree and determined the value of 60 as the best cut-off point. Also, we verified that the search performance was not affected significantly at this cut-off value (for all three methods). We use 60 as cut-off for all methods, except for SW-graph. In the case of SW-graph, we do not limit the maximum number of vertices degree since the SW-graph effectiveness depends mostly on their long edges, which probably would be deleted if applied the pruning.

4.3 GP set-up

The following GP configuration was used in our experiments. We based our choices on related work [1, 10, 30, 43] and empirical results:

- **Population**: for all three datasets, we created an initial population of 400 individuals, using the *ramped-half-and-half* technique, restricting the individuals' tree representation to a maximum depth of 5. The *ramped-half-and-half* [29] is a population initialization procedure widely used in the implementation of genetic programming solutions. In this technique half of the population is created using the Full method and the other half with the Grow method. The Full method generates random full trees, this is, trees where all leafs are at the same level. In the Grow case instead, this last property is not mandatory. Therefore, the trees are generated completely random.
- **Functions**: we employed the set of classical mathematical operators: $\{+, -, \times, /\}$. Additionally, we included the binary operators of max and min.
- **Terminals**: we included all the search dependent and independent properties described in Section 3: $\{D, L, E, N, C, J, P, A, R\}$. Also, to maintain these values approximately at the same scale, we divided them by their correspondent maximum values. We made this aiming to assign the same

[5] KGraph: `https://github.com/aaalgo/kgraph` (As of June 2021).
[6] NMSLIB: `https://github.com/nmslib/nmslib` (As of June 2021).

importance to all properties. Finally, we included random real values uniformly selected from the range $[-1, 1]$.
- **Genetic operations**: in all experiments, we used the classical operators of reproduction, mutation, and crossover, employing the tournament selection method, with size 6, as criteria to select the individuals. In all experiments, the reproduction, mutation, and crossover operators were applied at a rate of 5%, 10%, and 85% of the population, respectively.
- **Fitness function**: we employed the fitness function given by Equation 1. For the set $\mathcal{T}$, we selected the logarithmic scaled values of $\{10^2, 10^{2.1}, 10^{2.2}, ..., 10^{3.9}, 10^4\}$, guarantying that individuals are optimized for very distinct situations, ranging from a restricted number of distance calculations to less limited scenarios.
- **Stopping criterion**: in the parameter exploration phase, we observed that, in most of experiments, after the 100^{th} generation, the fitness value does not change significantly.

At fitness evaluation of individuals, a subset of 1,000 queries were selected randomly from the original test set of 10,000 queries, for each dataset. This subset compose the *training set* on the learning process. The remaining subset of 9,000 queries were used to test the best individual found through the evolution process, also known as the *test set*. It is worth mentioning that the queries on the training and test set are disjoint, so all queries on the test set were unseen on the training process.

4.4 Evaluation metric

We adopted the classical *Speedup vs Recall* charts to present the performance of each method. The speedup is measured over the linear search (explore all collection). To keep this comparison independent of the architecture where the experiments were executed, we considered the following definition of speedup [21]:

$$Speedup = \frac{Size\ of\ collection}{\#\ of\ distance\ calculations} \tag{2}$$

4.5 Experimental results

Results for 1-NN search (search of the one nearest point) of the proposed approach on the GloVe, YFCC, and SIFT datasets are presented in the first column of Fig. 8. Dashed curves (with suffix "GP") represent the search performance on the corresponding graphs using the GP-based scoring function for selecting the next vertex that will be explored.

For the GloVe dataset, as it can be observed, search performance results for the 1-NN case showed a significant gain obtained by using the GP-based scoring function against the usual distance. This gain is observed for all the four methods considered for NN graph construction. In the case of the YFCC

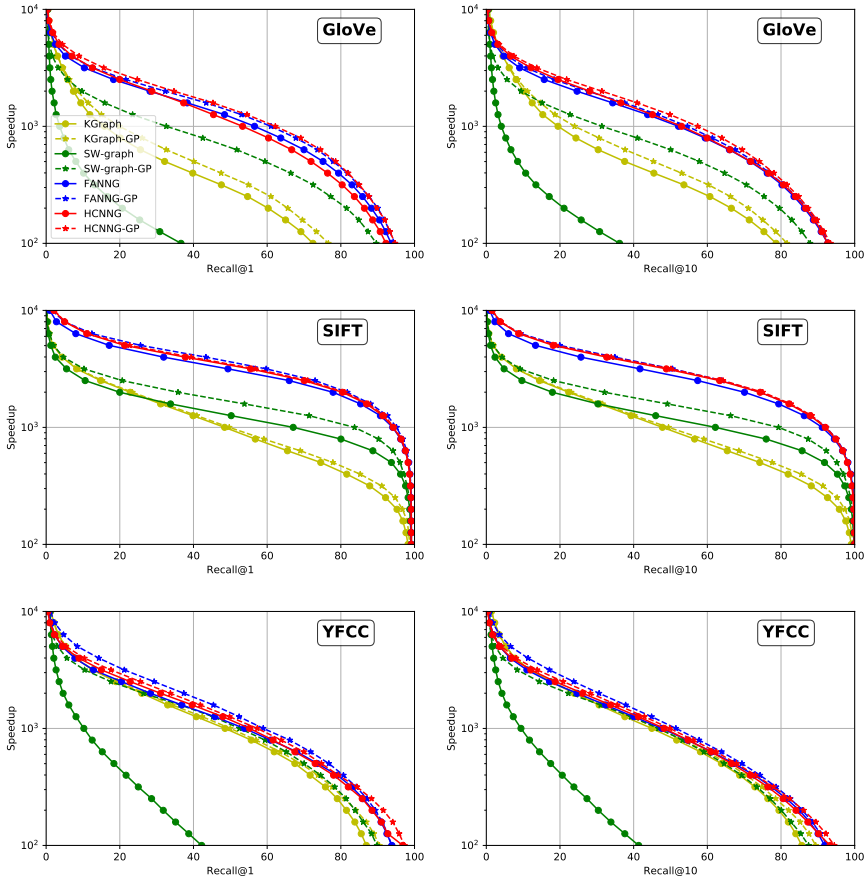

Fig. 8 Speedup vs recall on GloVe, SIFT, and YFCC datasets.

and SIFT datasets, note that, although, the margin between the two curves (baselines and proposed GP-based search) seems to be small, figures are shown in logarithmic scale. Also, we run statistical per-query paired t-test with 95% confidence, over the 9,000 queries employed as test queries, and considering the same values of speedup used in Fig. 8. Results of this statistical test are shown in Table 2, where "+" symbols means the statistical superiority of our GP-based search approach against classical search (Algorithm 1), symbol "−" means the opposite, and "=" means a statistical tie. The last row of the table presents the results of the statistical tests done over all queries and speedup values, considering all graph construction methods and datasets. These results demonstrate the statistical superiority of our approach against their corresponding baselines.

Employing the same functions discovered on the 1-NN search case, we also ran experiments for the 10-NN (search of the 10 nearest points). Results of

Table 2 Statistical paired t-test for 1-NN search, comparing our GP-based approach vs classical search ("+": gain, "−": lost, "=": tie, H: HCNNG, F: FANNG, S: SW-graph, K: KGraph).

Speedup	GloVe				SIFT				YFCC			
	H	F	S	K	H	F	S	K	H	F	S	K
$10^{2.0}$	+	+	+	+	=	=	+	+	+	=	+	+
$10^{2.1}$	+	+	+	+	=	−	+	+	+	=	+	+
$10^{2.2}$	+	+	+	+	=	−	+	+	+	+	+	+
$10^{2.3}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.4}$	+	+	+	+	=	=	+	+	+	+	+	+
$10^{2.5}$	+	+	+	+	=	=	+	+	+	+	+	+
$10^{2.6}$	+	+	+	+	=	=	+	+	+	+	+	+
$10^{2.7}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.8}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{2.9}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.0}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.1}$	+	+	+	+	=	+	+	+	+	+	+	+
$10^{3.2}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.3}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.4}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.5}$	+	+	+	+	+	+	+	+	+	+	+	=
$10^{3.6}$	+	+	+	=	+	+	+	+	+	+	+	+
$10^{3.7}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.8}$	+	+	+	−	+	+	+	+	+	+	+	=
$10^{3.9}$	+	+	+	=	+	+	+	+	+	+	+	+
$10^{4.0}$	+	+	+	=	=	+	+	+	+	+	+	=
General	+	+	+	+	+	+	+	+	+	+	+	+

these experiments are shown in the second column of Fig 8, for the GloVe, YFCC, and SIFT datasets. Similarly to the case of the 1-NN search, we also performed a statistical paired t-tests for the 10-NN search. These results are shown in Table 3. As it can be observed, the behavior of all methods did not change significantly. The proposed approach yielded superior or equivalent results to their counterparts, in almost all points, demonstrating the effectiveness of discovered functions in the scenario of K-NN searches, even though these were trained to optimize the 1-NN search.

The following scoring functions were discovered by the proposed GP framework and employed in the final experiments that led to the results described above:

- **GloVe**:
  - **HCNNG**: $min((E+max(A,C)),(D+D))/(0.68+E)+min((((D/L)/E)\times(D+min(D,L))),((min(D,P)/0.68)+min(0.68,D)))$
  - **FANNG**: $min(L,min(D,(L+N)))/(((L\times min(D,E))\times(L+(-0.94\times D)))+((((D-E)-L)/(D\times L))+((N-0.94\times J)+(N+min(C,P))))$
  - **SW-graph**: $(min(min(D,J),min(A,C))+min(C,D))-max(min(C,D),((D+N)\times(C+E)))+max((min(A,J)\times min(D,N)),min(C,J)+(D+N))+(min(min(D,N),min(C,J))+(D+min(C,J)))$
  - **KGraph**: $(max((N-E)-E,max(D,P)-max(D,J))\times min(D,(P-max(0.96,D))))\times(max(((0.96-E)-E),0.96-L)+(((R-D)-D)\times((R-E)-max(E,P))))$
- **YFCC**:
  - **HCNNG**: $max((max(A,N)\times(P+(D/L))),(-0.11+(L\times(E\times L))))+((-0.11/min(D,(A+D)))+max((-0.11/min(D,N)),max(E,P)))$
  - **FANNG**: $max(min(min(C,J),max(0.10,J)\times min(L,N)),(N\times min(D,P)+(min(0.61,D)+D)))+min(min(0.61\times E,(D\times N)+(N\times N)),min(min(D,min(D,N)),min(D,(0.61\times N))))$
  - **SW-graph**: $((D/(-0.45+N))\times(L\times N))+(max(N,C\times N)+max(D,0.84\times N))-min(min((0.84\times N)/(-0.45+J),min(C+N,D+E)),((L\times N)/(-0.45+D)-min(0.84\times N,C+N)))$
  - **KGraph**: $min(((((-0.13+E)/D)-(D+(-0.52\times N)))-(E+(P+0.13))/D),((min(A,N)+(N-R))-(min(A,R)/E))-((E+(P+0.13))/D))$

Table 3 Statistical paired t-test for 10-NN search, comparing our GP-based approach vs classical search ("+": gain, "−": lost, "=": tie, H: HCNNG, F: FANNG, S: SW-graph, K: KGraph).

Speedup	GloVe				SIFT				YFCC			
	H	F	S	K	H	F	S	K	H	F	S	K
$10^{2.0}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.1}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.2}$	+	+	+	+	+	−	+	+	+	+	+	+
$10^{2.3}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.4}$	+	+	+	+	+	−	+	+	+	+	+	+
$10^{2.5}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.6}$	+	+	+	+	+	=	+	+	+	+	+	+
$10^{2.7}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{2.8}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{2.9}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.0}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.1}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.2}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.3}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.4}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.5}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.6}$	+	+	+	=	+	+	+	+	+	+	+	=
$10^{3.7}$	+	+	+	=	+	+	+	+	+	+	+	=
$10^{3.8}$	+	+	+	−	+	+	+	+	+	+	+	=
$10^{3.9}$	+	+	+	=	+	+	+	+	+	+	+	=
$10^{4.0}$	+	+	+	−	+	+	+	+	+	+	+	=
General	+	+	+	+	+	+	+	+	+	+	+	+

- **SIFT**:
  - **HCNNG**: $min(((min(0.42, P)\times(A+D))\times(E-(0.42/D))), (N-(0.42/D)))+(((min(0.42, P) \times (A + E)) \times (min(0.42, D) + (A + D))) - (E + min(D, min(D, P))))$
  - **FANNG**: $((((C/L) + min(N, P)) - (J - max(E, N))) + (((-0.64/D) + max(C, N)) - (min(P, P) - max(N, P)))) - ((min(P, (C/D)) - max(max(N, P), max(C, D))) - max((-0.64 \times (-0.64/D)), ((D/P) + max(L, N))))$
  - **SW-graph**: $(N \times (((D/E) \times max(N, P)) + ((D/N) \times max(D, N)))) + max((max(E, N) \times (D + N)), (D + min(J, P))) + ((D + min(P, R)) \times (D + (N + P)))$
  - **KGraph**: $min((D-(min(D, E)\times min(0.41, E))), (max(min(C, J), (E-C)) + min(max(E, J), max(0.41, E)))) - (min(max(min(0.41, J), (E - C)), max((R/E), min(C, D))) \times (min(J, (D/R)) \times (max(0.41, R) + min(0.41, E))))$

In order to evaluate the influence of each property on each pair of dataset and NN graph technique, we performed a set of experiments comparing the 1-NN search results obtained by using the GP-based functions listed previously against the results obtained by using the same functions but with each property removed from the expression, one at a time. Each removed property was replaced by a neutral value (either 0 or 1, depending on the math function in which it was involved).

We also performed per-query paired t-tests with 95% confidence comparing these two approaches. These results are shown in Table 4, where "−" means that the exclusion of the property affected negatively the search results, "+" the opposite, and "=" means that the exclusion of the property did not affect significantly the search performance. The cases where the property was not present in the GP-based function are shown as blank cells. As it can be observed, the distance (D) and the edge weight (E) are the most employed properties in the discovered functions, and the exclusion of these properties affected negatively the search performance in almost all cases, so, they were among the most important properties. Also, there were a few cases (e.g., vertex degree (N) and Jaccard coefficient (J)) for which the exclusion of some prop-

Table 4 Statistical paired t-test for 1-NN search, comparing our GP-based approach using original functions vs using the resulting function by excluding each property ("+": gain, "−": lost, "=": tie, H: HCNNG, F: FANNG, S: SW-graph, K: KGraph). The cases where the property was not present in the GP-based function are shown as blank cells.

	GLOVE				YFCC				SIFT			
Property	H	F	S	K	H	F	S	K	H	F	S	K
D	−	−	−	−	−	−	−	−	−	−	−	−
L	−	−		−	=		+			=		
E	−	−	=	−	−	−	=	−	=	−	−	−
N		=	+	−	−	+	−	+	−	−	+	
C	=	=	−				=			−		−
J		+	−	−			=			=	+	−
P	=	=		−	−	−		−	=	=	−	
A	=		=		−	=		=	=			
R				−		−		−			−	−

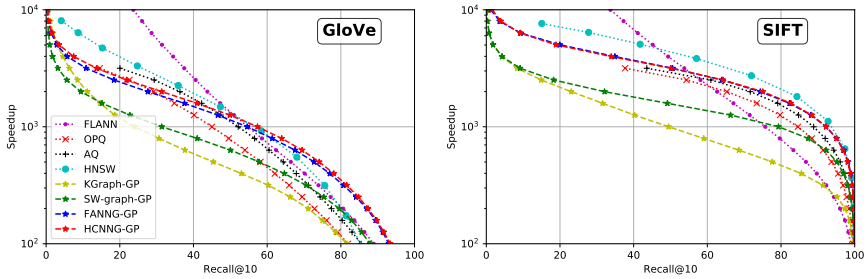

Fig. 9 Comparison with the state-of-the-art techniques.

erties increased significantly the search performance, therefore, this analysis can be used in the future for proper property selection.

4.5.1 Comparison with state of the art

Fig. 9 compares the graph-based methods investigated in this paper with others belonging to different families (tree-based and quantization-based) for the GloVe and SIFT datasets, considering the speedup vs Recall@10. The methods compared are FLANN [38], OPQ [16], and AQ [6]. Also, we included in this comparison the HNSW described above, since it presented top performances in recent benchmarks. As we can observe, from the graph-based methods in which we employ our search approach, FANNG and HCNNG obtained the best search performance. On the other hand,with regard to other state-of-the-art techniques, HNSW is the most competitive for these datasets. That suggests that the investigation of the use of a GP formulation in a hierarchical structure is a promising research direction. We expect to investigate that in future work.

4.5.2 Search time vs recall

Fig. 10 shows the speedup in terms of raw time considering all evaluated methods and datasets (GloVe, SIFT, and YFCC). In the chart, *time* refers to the

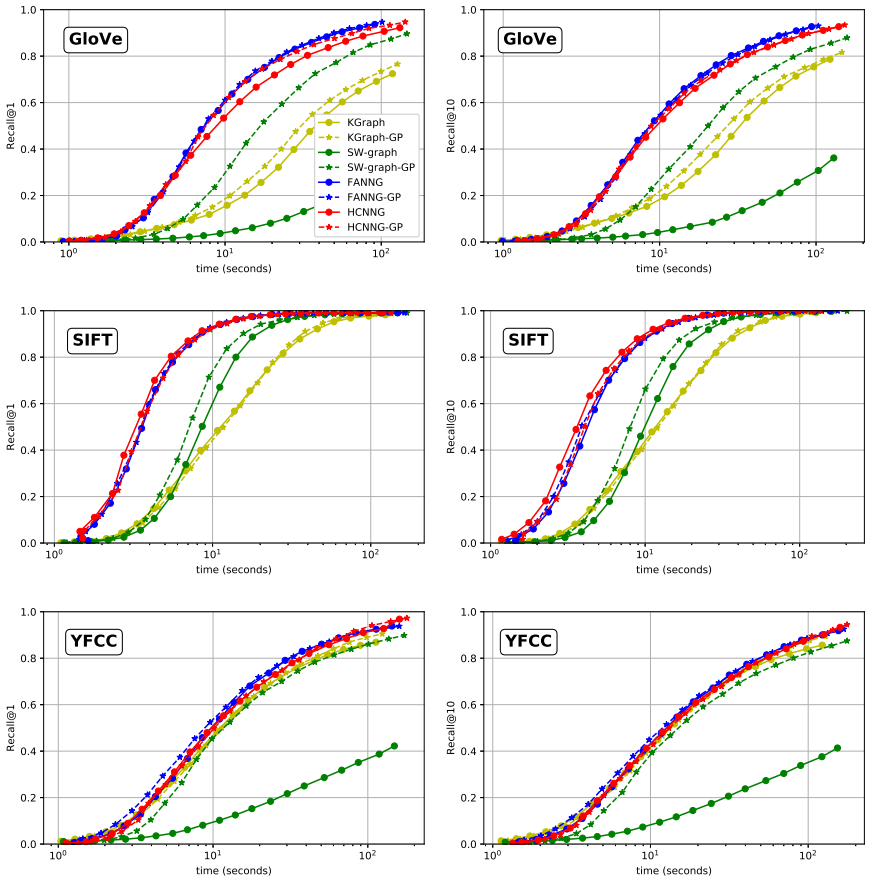

Fig. 10 Time vs recall on GloVe, SIFT, and YFCC datasets.

total time for processing 9000 queries. As we can observe, despite the use of a complex function for guiding the selection of suitable paths in a search, the overall gains regarding the use of the GP-based approach led to better or comparable results, especially for the GloVe dataset. One exception, for example, was the performance of HCNNG with and without GP (red curves) for the SIFT dataset. In this case, the use of GP led to an overhead. Search results in terms of effectiveness in this dataset are high in general for all methods, in comparison to the other datasets, leading therefore to small margins for improvements.

5 Conclusions

We introduced in this work a genetic programming framework that explores different topological properties in NN graphs, in order to improve the order in

which vertices are visited at search time, and, therefore, reduce the number of vertices explored to discover the true nearest neighbors. Our experimental results and statistical tests showed that searches on NN graphs can be improved by considering other kinds of vertices' properties besides the distance to the query, combining them by means of the mathematical expression returned by the learning stage of the proposed framework. Also, the proposed technique led to improvements against the usual search approach, in terms of recall, independently of the four NN graph techniques employed, and with no significant time overhead.

In scenarios with very large collections, in the order of dozens of millions of objects, where common computers are not capable of executing NN graph-based methods due to memory constrains, it is possible to employ a distributed version of these algorithms by applying for example the MapReduce programming model. Some distributed approaches were proposed in the literature [18,38] for the tree-based and hash-based methods. Commonly, data is split across various computers, where indices are created separately. Then, search is conduced independently on each machine and results are merged. This approach can be easily adapted for NN graph-based algorithms as for our proposed framework. We intend, as future work, to experiment with a distributed version of our framework.

Also, we plan to include other topological properties in the proposed framework that could improve even more our results. We also plan to deepen the investigation of the impact of using different graph measurements in the GP-based combination discovery framework. Finally, we intend to apply the proposed framework in other contexts that might benefit from NN graphs, e.g., image annotation problems.

Acknowledgements Authors are grateful to CNPq (grants #307560/2016-3, #400487/2016-0, #425340/2016-3, #304380/2018-0, and #422593/2018-4), CAPES (grant #88881.145912/2017-01), FAPESP (grants #2014/12236-1, #2015/24494-8, #2016/50250-1, #2017/20945-0, #2015/11937-9, #2017/12646-3, and #2017/16246-0) and the FAPESP-Microsoft Virtual Institute (grants #2013/50155-0, #2013/50169-1, and #2014/50715-9). This study was financed in part by the Coordenação de Aperfeiçoa-mento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors are also grateful to the NFR SmartPlan and TwinFjord projects.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Albarracín, J.F.H., Ferreira, E., dos Santos, J.A., da S. Torres, R.: Fusion of genetic-programming-based indices in hyperspectral image classification tasks. In: Proceeding of the IEEE International Geoscience and Remote Sensing Symposium, pp. 554–557 (2017)

2. Albarracín, J.F.H., Oliveira, R., Hirota, M., dos Santos, J.A., da S. Torres, R.: A soft computing approach for selecting and combining spectral bands. Remote Sensing **12**(4), 2267 (2020). DOI 10.3390/rs12142267. URL https://www.mdpi.com/769804
3. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM **51**(1), 117–122 (2008)
4. André, F., Kermarrec, A.M., Le Scouarnec, N.: Accelerated nearest neighbor search with quick adc. In: Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, ICMR'2017, pp. 159–166. ACM, New York, NY, USA (2017)
5. Aumüller, M., Bernhardsson, E., Faithfull, A.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In: C. Beecks, F. Borutta, P. Kröger, T. Seidl (eds.) Similarity Search and Applications, pp. 34–49. Springer International Publishing, Cham (2017)
6. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: Proceedings of the Conference on Computer Vision and Pattern Recognition, pp. 931–938 (2014)
7. Babenko, A., Lempitsky, V.: The inverted multi-index. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(6), 1247–1260 (2015)
8. Bawa, M., Condie, T., Ganesan, P.: LSH forest: Self-tuning indexes for similarity search. In: Proceedings of the 14th International Conference on World Wide Web, WWW'2005, pp. 651–660 (2005)
9. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)
10. de Carvalho, M.G., Laender, A.H.F., Gonçalves, M.A., da Silva, A.S.: A genetic programming approach to record deduplication. IEEE Transactions on Knowledge and Data Engineering **24**(3), 399–412 (2012)
11. Chiu, C.Y., Chiu, J.S., Markchit, S., Chou, S.H.: Effective product quantization-based indexing for nearest neighbor search. Multimedia Tools and Applications **78**(3), 2877–2895 (2019)
12. Costa, L.d.F., Rodrigues, F.A., Travieso, G., Villas Boas, P.R.: Characterization of complex networks: A survey of measurements. Advances in physics **56**(1), 167–242 (2007)
13. Dasgupta, S., Freund, Y.: Random projection trees and low dimensional manifolds. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC'2008, pp. 537–546 (2008)
14. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web, WWW'2011, pp. 577–586 (2011)
15. Efstathiades, C., Efentakis, A., Pfoser, D.: Efficient processing of relevant nearest-neighbor queries. ACM Transactions on Spatial Algorithms and Systems **2**(3) (2016)
16. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2946–2953 (2013)
17. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A* search meets graph theory. In: SODA'2005 (2005)
18. Gonzalez-Lopez, J., Ventura, S., Cano, A.: Distributed nearest neighbor classification for large-scale multi-label data on spark. Future Generation Computer Systems **87**, 66–82 (2018)
19. Gubichev, A., Bedathur, S., Seufert, S., Weikum, G.: Fast and accurate estimation of shortest paths in large graphs. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM'2010, p. 499–508. Association for Computing Machinery, New York, NY, USA (2010)
20. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD'1984, pp. 47–57. ACM, New York, NY, USA (1984)
21. Harwood, B., Drummond, T.: FANNG: Fast approximate nearest neighbour graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5713–5722 (2016)

22. Houle, M.E., Ma, X., Oria, V., Sun, J.: Improving the quality of k-nn graphs for image databases through vector sparsification. In: Proceedings of International Conference on Multimedia Retrieval, ICMR'2014, pp. 89:89–89:96. ACM (2014)
23. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 604–613 (1998)
24. Iwasaki, M.: Pruned bi-directed k-nearest neighbor graph for proximity search. In: Similarity Search and Applications, pp. 20–33. Springer International Publishing, Cham (2016)
25. Jebari, K.: Selection methods for genetic algorithms. International Journal of Emerging Sciences **3**, 333–344 (2013)
26. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(1), 117–128 (2011)
27. Ji, T., Liu, X., Deng, C., Huang, L., Lang, B.: Query-adaptive hash code ranking for fast nearest neighbor search. In: Proceedings of the 22nd ACM International Conference on Multimedia, MM'2014, pp. 1005–1008. ACM, New York, NY, USA (2014)
28. Kalantidis, Y., Avrithis, Y.: Locally optimized product quantization for approximate nearest neighbor search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2329–2336 (2014)
29. Koza, J.R.: Genetic Programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
30. Lacerda, A., Cristo, M., Gonçalves, M.A., Fan, W., Ziviani, N., Ribeiro-Neto, B.: Learning to advertise. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'2006, pp. 549–556. ACM, New York, NY, USA (2006)
31. Li, J., Lan, X., Wang, J., Yang, M., Zheng, N.: Fast additive quantization for vector compression in nearest neighbor search. Multimedia Tools and Applications **76**(22), 23273–23289 (2017)
32. Liu, J., Li, M., Liu, Q., Lu, H., Ma, S.: Image annotation via graph learning. Pattern Recognition **42**(2), 218–228 (2009)
33. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 2, pp. 1150–1157 vol.2 (1999). DOI 10.1109/ICCV.1999.790410
34. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB'2007, pp. 950–961 (2007)
35. Malkov, Y.A., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems **45**, 61–68 (2014)
36. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. arXiv preprint arXiv:1603.09320 (2016)
37. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: Proceedings of the International Conference on Computer Vision Theory and Application, pp. 331–340 (2009)
38. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(11), 2227–2240 (2014)
39. Papadias, D.: Hill climbing algorithms for content-based retrieval of similar configurations. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'2000, pp. 240–247. ACM, New York, NY, USA (2000)
40. Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1532–1543 (2014)
41. Popescu, A., Spyromitros-Xioufis, E., Papadopoulos, S., Le Borgne, H., Kompatsiaris, I.: Toward an automatic evaluation of retrieval performance with large scale image collections. In: Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions, MMCommons'2015, pp. 7–12. ACM, New York, NY, USA (2015)

42. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: Proceedings of the 18th ACM conference on Information and knowledge management, CIKM'2009, pp. 867–876 (2009)
43. da S. Torres, R., Falcão, A.X., Gonçalves, M.A., Papa, J.P., Zhang, B., Fan, W., Fox, E.A.: A genetic programming framework for content-based image retrieval. Pattern Recognition **42**(2), 283–292 (2009)
44. Silpa-Anan, C., Hartley, R.: Optimised kd-trees for fast image descriptor matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
45. Sproull, R.F.: Refinements to nearest-neighbor searching in k-dimensional trees. Algorithmica **6**(1-6), 579–589 (1991)
46. Su, F., Xue, L.: Graph learning on k nearest neighbours for automatic image annotation. In: Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, ICMR'2015, pp. 403–410. ACM, New York, NY, USA (2015)
47. Tang, J., Hong, R., Yan, S., Chua, T.S., Qi, G.J., Jain, R.: Image annotation by knn-sparse graph-based label propagation over noisily tagged web images. ACM Transactions on Intelligent Systems and Technology **2**(2), 14:1–14:15 (2011)
48. Tretyakov, K., Armas-Cervantes, A., García-Bañuelos, L., Vilo, J., Dumas, M.: Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM'2011, p. 1785–1794. Association for Computing Machinery, New York, NY, USA (2011)
49. Vargas, J.A.: Large-scale indexing of high dimensional data via nearest neighbors graphs. Ph.D. thesis, University of Campinas, São Paulo, Brazil (2020)
50. Vargas, J.A., Dias, Z., da S. Torres, R.: A genetic programming approach for searching on nearest neighbors graphs. In: Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR '19, pp. 43–47. ACM, New York, NY, USA (2019)
51. Vargas, J.A., Gonçalves, M.A., Dias, Z., da S. Torres, R.: Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. Pattern Recognition **96**, 106970 (2019)
52. Wang, M., Zhou, W., Tian, Q., Pu, J., Li, H.: Deep supervised quantization by self-organizing map. In: Proceedings of the 25th ACM International Conference on Multimedia, MM'2017, pp. 1707–1715. ACM, New York, NY, USA (2017)
53. Xiaokang, F., Jiangtao, C., Hui, L., Yingfan, L.: An efficient lsh indexing on discriminative short codes for high-dimensional nearest neighbors. Multimedia Tools and Applications **78**(17), 24407–24429 (2019)
54. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 311–321 (1993)