UNIVERSITÀ DEGLI STUDI DI TRIESTE

Towards Proximity Graph Auto-configuration: An Approach Based on Meta-learning

This is the peer reviewed (post-print) version of the following article:

*Original*
Towards Proximity Graph Auto-configuration: An Approach Based on Meta-learning / Seidi Oyamada, R., Shimomura, R.C., Barbon Junior, S., Kaster, D.S.. - 12245:(2020), pp. 93-107. (24th European Conference on Advances in Databases and Information Systems,ADBIS 2020 fra 2020) [10.1007/978-3-030-54832-2_9].

*Availability:*
This version is available at: 11368/3037311 since: 2023-01-03T11:51:29Z

*Publisher:*
Springer

*Published*
DOI:10.1007/978-3-030-54832-2_9

*Terms of use:*

Some rights reserved. More details in the PostPrint rights section below.

*License:*
Digital Rights Management not defined

*Publisher copyright*

-.-

(Article begins on next page)

22 October 2026

arTs Università degli Studi di Trieste
Archivio della ricerca – postprint

# Towards Proximity Graph Auto-configuration: An Approach Based on Meta-learning

Rafael Seidi Oyamada[1(✉)], Larissa C. Shimomura[2], Sylvio Barbon Junior[1], and Daniel S. Kaster[1]

[1] State University of Londrina, Londrina, PR, Brazil
{rseidi.oyamada,barbon,dskaster}@uel.br
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
l.capobianco.shimomura@tue.nl

**Abstract.** Due to the high production of complex data, the last decades have provided a huge advance in the development of similarity search methods. Recently graph-based methods have outperformed other ones in the literature of approximate similarity search. However, a graph employed on a dataset may present different behaviors depending on its parameters. Therefore, finding a suitable graph configuration is a time-consuming task, due to the necessity to build a structure for each parameterization. Our main contribution is to save time avoiding this exhaustive process. We propose in this work an intelligent approach based on meta-learning techniques to recommend a suitable graph along with its set of parameters for a given dataset. We also present and evaluate generic and tuned instantiations of the approach using Random Forests as the meta-model. The experiments reveal that our approach is able to perform high quality recommendations based on the user preferences.

**Keywords:** Proximity graphs · Nearest neighbor search · Meta-learning · Auto configuration

## 1 Introduction

Dealing with complex data (images, long texts, audios, and etc.) is a typical task in different application areas, such as pattern recognition, image retrieval, data mining, etc. In general, complex data is represented through feature vectors composed of measures and properties extracted from the intrinsic content of the data and retrieved using dissimilarity relations between pairs of feature vectors. Those are known as *similarity queries*, as they retrieve the elements from the dataset that satisfy a given similarity-based criterion, such as the $k$-Nearest

This work has been supported by CAPES and CNPq funding agencies, and also has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825041.

© Springer Nature Switzerland AG 2020
J. Darmont et al. (Eds.): ADBIS 2020, LNCS 12245, pp. 93–107, 2020.
https://doi.org/10.1007/978-3-030-54832-2_9

Neighbors query ($k$-$NNq$), which retrieves the $k$ most similar elements the query element [20].

There are several access methods suitable for indexing complex data in the literature. These methods can be divided into four groups: tree-based [13], hashing-based [12], permutation-based [2], and *graph-based* [24]. In this paper, we focus on graph-based methods as recent works have shown that this method type has often outperformed other methods types in approximate similarity search [10,17,19].

Graph-based methods are very sensitive to user-defined parameters for both construction and querying. The graph structure is mainly affected by the number of neighbors an element (vertex) should be connected to. More edges in the graph generate shorter paths to be traversed. However, more edges increase the memory footprint, as well as the cost of vertex expansion during the search, since the adjacency of each vertex is larger. Additionally, depending on the graph structure, the query algorithm may not be able to find a path from the query element to every element that is part of the answer. A common approach to alleviate this problem is to execute a parameter defined number of traversals each of which is starting from a different source vertex. The number of traversals, or restarts, is also a sensible parameter as it allows improving the result accuracy at the cost of degrading the execution time. Setting suitable values for these parameters is hard as they depend on several factors, including the type of graph-based method, the dataset properties and the optimization goal (e.g., query time or memory requirements).

Recent works showed that there are no default parameters for graph-based methods (see [24] and the references therein). Considering the difficulty to reproduce experimental results for approximate nearest neighbor (ANN) algorithms in general, Aumüller et al. [3] proposed an automated benchmarking system for evaluating existing tree-based, hash-based, and graph-based methods. The authors performed experiments on different datasets so that future users can use it as a starting point for their applications. In our previous work, we performed a deep behavior analysis of graph-based methods given its settings, regarding several metrics, such as search and construction time, recall, and memory usage [24]. Our results indicated that it is not possible to assert that there is a better type of graph for all cases. Although these works identified general patterns that are useful to guide the choice of the type of graph and its parameters, it is hard to find good parametrizations for a variety of cases. Usually, a grid search procedure is employed to define a suitable graph configuration. However, this is time consuming and limited to the tested combinations.

In this paper, we propose a machine learning approach to recommend a suitable graph-based method as well as its main parameters for a given dataset and similarity query requirements. Our approach employs meta-learning techniques for providing high-performance configurations for each graph-based method by examining its dataset pattern. This work presents an instantiation of this approach for recommending parameters for dimensional datasets. The dataset characterization includes descriptions such as embedding and intrinsic dimensionality, cardinality, and statistical and information-theoretical measures. The prediction targets include query quality (in terms of query recall) and execution

time, both measured using different types of graph-based methods over a collection of real and synthetic datasets. We trained a meta-model to predict the performance of the graph-based methods according to the dataset properties and query requirements to quickly evaluate characterizations to recommend the best predicted one. Our results using a Random Forest regressor have achieved a high-quality recommendation for most situations, which means that our proposal is able to generalize datasets to provide suitable configurations for graph-based methods for similarity retrieval of image databases.

This work is organized as follows. Section 2 describes the problem of parameter setting for graph-based methods for similarity searches as well as related works. Section 3 presents our approach based on meta-learning, and Sect. 4 presents the experimental evaluation and results. Lastly, on Sect. 5, we present our conclusion and future works.

## 2 Parameter Setting for Graph-Based Indexing Methods

The most common type of graph used for similarity searches is the proximity graph [24]. A proximity graph is a graph in which each pair of vertices $(v, u) \in V$ is connected by an edge $e = (u, v)$, $e \in E$, if and only if $u$ and $v$ satisfy a given property $P$, called neighborhood criterion, which defines the type of the graph.

Among the graph-based methods for similarity searches, the $k$-Nearest Neighbor Graph ($k$-*NNG*) [10,21] and the navigable small-world graph (*NSW*) [17] are two important types of graphs. The $k$-*NNG* has well-known properties that are useful for performing similarity searches and has been used as the base for several other graph-based methods. The brute-force construction of the $k$-*NNG* has a quadratic computational cost, other construction algorithms with lower cost have been proposed in the literature [22]. These construction algorithms generate an approximated version of the actual $k$-*NN* graph in a shorter time than the brute-force construction. One remarkable method is the *NN-Descent* [8] in which the main algorithm idea is "the neighbor of a neighbor is probably a neighbor". The Navigable Small World graph (*NSW*) is a recent proposal that is also based on connecting elements to their nearest neighbors, however, it uses short- and long-range undirected edges that grant the graph small-world properties [17]. The main advantages of the *NSW* are fast and highly precise approximate search execution thanks to the small-world properties, and its fast construction algorithm. Both the $k$-*NNG* and the *NSW* are sensible to construction parameters, particularly the number of neighbors of each vertex (*NN*)[1], which defines the number of edges in the graph. The parameter *NN* impacts both the query quality and the execution time since it affects the number and length of paths in the graph as well as the cost of evaluating the adjacency of each vertex.

There are different strategies to search for similar data in proximity graphs. The fundamental approach is to use spatial approximation, introduced by [20].

[1] In this paper we use *NN* to define the construction parameter number of neighbors of the graph-based methods, and $k$ to define the query parameter number of neighbors in a $k$-*NN* query.

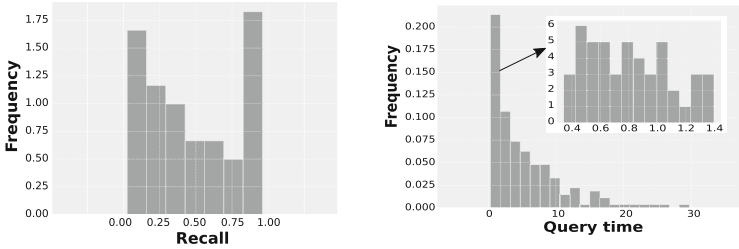
The spatial approximation property allows, starting the search from a source vertex traverse iteratively the graph using greedy steps to get spatially closer and closer to objects that are most similar to the query element. The Graph Nearest Neighbor Search (*GNNS*) is an effective algorithm that executes multiple greedy searches based on spatial approximation and aggregates the partial results into the final result [10]. In the *GNNS*, the multiple searches are called *restarts* ($R$), whose number is a user-defined parameter. The $R$ parameter allows improving the quality of the result as each search starts from a different source and traverses a different path in the graph. Nevertheless, the number of restarts also impacts query execution time.

### 2.1 The Impact of Parameters for Graph-Based Methods

This section shows that defining suitable values for these parameters has a major impact on the effectiveness and performance of the methods and is a challenging problem. Our discussion considers typical scenarios.

The first scenario states that the user makes a careless choice and uses the same configuration across different datasets. This scenario usually happens when a configuration provides a good result for a given dataset. Then, for simplicity, the user replicates the "good" configuration for other datasets. To illustrate this scenario, we fixed the configuration, built a graph-based method for different datasets, executed *k-NN* queries using these indices, and analyzed the results. We ran this test using several configurations varying the graph type, and the construction and query parameters. Figure 1(a) shows results of a representative example, which corresponds to a *NN-Descent* graph set with $NN = 25$ running *k-NN* queries with $k = 30$ using the *GNNS* algorithm with $R = 10$ for all datasets used in this work (see details in Subsect. 3.1). The figure shows the distribution of the average recall rates throughout all datasets, being the recall rate for each dataset computed as the average recall for 100 queries with random query elements. The recall rate of a query is the fraction of the true $k$-nearest neighbors to the query element that is retrieved by the query. It is noticeable that the same set of parameters for distinct datasets leads to completely different quality rates for queries. Similar reasoning is also valid regarding query time.

The second scenario considers a single dataset. In this scenario, the user has to set the ideal parameters for the dataset subject to some constraints. Figure 1(b) presents the distribution of the average execution time for 30-*NN* queries using the *GNNS* search in the *NN-Descent*, considering different parameters, for the dataset *Color Histogram*, whose features are the 32-bin color histogram of a set of 68,040 images. The goal here is only to define the parameters $NN$ and $R$ for the *NN-Descent* for this dataset. The constraint is that the query should have a recall of at least 0.95. Analyzing the query time distribution, we can notice that almost two-thirds of the tested combinations of $NN$ and $R$ do not lie the first bucket, which means that the execution time is at least twice larger than the time demanded by the best configurations. The figure also shows the histogram of the configurations that lie in the first bucket. We can see that the variance regarding the average execution time is also large for the best configurations.




(a) (b)

**Fig. 1.** Distribution of (a) recall rates given a fixed configuration over several datasets and (b) query times with recall > 0.95 varying configurations.

Additionally, regarding only the top-15 configurations, the methods present a variation of up to 50% in the execution time, which is significant.

The third scenario is the more complete one as it requires to choose the best graph type and its configuration for a given dataset. Figure 2 shows how the graph types *NSW* and *k-NNG* behave for increasing values for the *NN* parameter for the dataset Texture, which has texture features of 68,040 images. The figure shows the results for the *k-NNG* built using two construction algorithms: *NN-Descent* and brute-force (*Brute-kNNG*). Each point in the plots corresponds to the smallest number of restarts and, consequently, the smallest query time, for the corresponding type of graph and *NN* value that returned results subject to the constraint of having a recall of at least 0.95. Analyzing the plots, if the optimization goal is memory (i.e., the configuration that satisfies the constraint that consumes less memory), the best option is the *NSW* with $NN = 5$. On the other hand, if the optimization goal is query time (i.e., the fastest configuration that satisfies the constraint), we have two configurations that tie: *NSW* with $NN = 100$, and *NN-Descent* with $NN = 130$, being the latter the best cost-benefit option as it demands less memory than the former option. We can also see that choosing the graph type that is the fastest in general, which is the *NSW* in this case, but with a poor configuration (e.g., $NN = 70$), may be the worst option among the graph types. The opposite is also true since the *Brute-kNNG* is the slowest method in general, however, it is the fastest for $NN = 150$. Finally, the plots indicate that every method has an optimal configuration, which varies for different datasets and constraints. All of these reasons reinforce that the problem of recommending optimal parameters to configure graph-based methods is important and challenging.

## 2.2 Related Work

Works in the literature of similarity searches have defined such parameters based either on the user intuition or exhaustive evaluation. These approaches lead to suboptimal configurations and/or are excessively time consuming since the identification of adequate parameters is a challenging problem. Some works in the

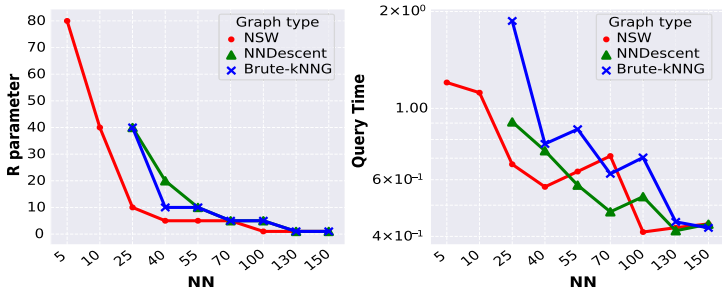



**Fig. 2.** a) Smallest number of restarts for each graph and *NN* value. b) Query time for the corresponding configurations of the plot on the left.

literature present extensive evaluations of access methods for similarity search. For instance, Li et al. performed a comprehensive experimental study on Approximate Nearest Neighbor (ANN) algorithms to provide a better understanding of their general behavior [16]. Similarly, the *ANN-Benchmark* was proposed to standardize the evaluation among ANN algorithms [3]. This benchmark works as a tool that enables comparing a wide range of ANN algorithms and its configuration over several real datasets. Specifically for graph-based methods, we performed an experimental evaluation in a previous work [24]. In this evaluation, we showed relevant trade-offs in the behavior of each graph according to several construction and search parameters. Although the performance patterns presented in these works can be useful, it is clear the difficulty of finding the ideal set of parameters for each algorithm considering the trade-offs among metrics, such as query time, and memory usage.

In the literature, there are works making use of machine learning concepts to advancing the state-of-the-art in database management [5,14]. However, none of them addresses parameter auto-configuration or algorithm selection. To the best of our knowledge, the only method of auto-selecting a suitable algorithm configuration for similarity searches was proposed by Muja and Lowe [18]. Their purpose is to minimize a cost function based on query time, indexing time, and memory usage. This is performed in two steps: first, a grid search strategy is used to find the parameter values that minimize the function; subsequently, a local exploration is realized to fine-tune the parameters obtained. A limitation of this work is the fact it only accepts tree-based methods.

Algorithm and parameter recommendation for machine-learning methods is an active research topic, and many successful approaches have relied on meta-learning [1,25]. Meta-learning differs from traditional learning methods (a.k.a. base-learning) in the notion of what to learn. Instead of producing a predictive function over a single problem domain, meta-learning attempts to gather knowledge from several domains to find patterns among them and provide suitable solutions for future problems. We can formally define meta-learning as follows [26]. Considering a task $t_i \in T$ (set of all tasks) along with its algorithm configuration $\theta_j \in \Theta$ (configuration space), we have a set of evaluations $P$, where

$P_{ij} = P(t_i, \theta_j)$ is the task $t_i$ solved by the algorithm configuration $\theta_j$ according to a performance metric, e.g. accuracy or recall. The evaluations $P$ refer to meta-instances, which are described through meta-features with the properties of the data enabling learning algorithms to find patterns among them, and the performances obtained by the evaluated algorithms/configurations in the different cases are the corresponding meta-targets. The meta-dataset is the set of meta-instances. A meta-model can be induced through $P$ using the meta-dataset to predict or recommend a suitable algorithm configuration for a given new task. In this context, the novelty of our proposal is to apply meta-learning to recommend configurations of graph-based methods for similarity search.

## 3 A Meta-learning Approach for Proximity Graph Parameter Recommendation

This section presents a proposal of an intelligent system to recommend suitable configurations for proximity graphs, given a dataset, the optimization goal, and the search properties and constraints. Our approach employs meta-learning to induce regression meta-models able to predict the performance of query executions grounded on graph-based methods from complex data. A representative collection of datasets and diverse configurations of graph-based methods was applied to obtain a robust and general regression meta-model.

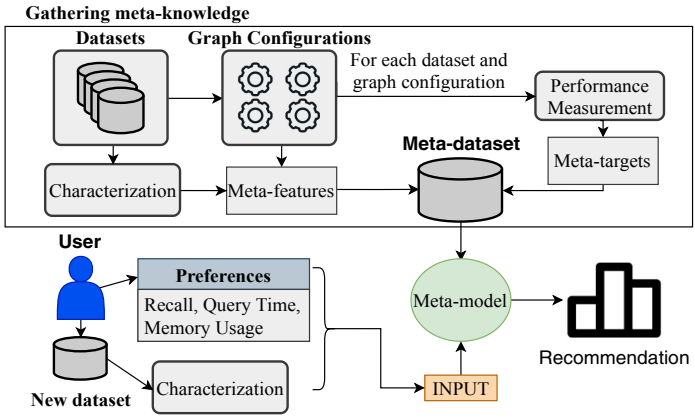



**Fig. 3.** Induction and usage of our proposed meta-learning recommender.

Figure 3 summarizes the process of the proposed recommender. It illustrates the steps of gathering meta-knowledge, inducting the meta-model, and generating the recommendations. The input is a set of datasets, a set of graph-based methods, and a set of parameter values. For each input dataset, we do a simple data augmentation by generating sub-datasets of smaller cardinalities and adding

them to the set of datasets. The set of graph configurations are combinations of graph type and values for the considered parameters. For each dataset in the set of datasets, the recommender generates meta-instances that are composed of meta-features extracted from the dataset (Characterization) and a graph configuration. The meta-dataset is generated by meta-instances that associate the meta-features to meta-targets. A meta-target is a measure (e.g., average query time or recall) obtained by running batteries of queries using the graph-based method built for the dataset using the meta-instance's construction and query configuration values (Performance Measurement).

Then, the recommender applies a meta-learner to induce a meta-model, which is one or more regressors trained using the meta-dataset. To obtain a recommendation, a user provides a new dataset and some requirements, which include the constraints to be satisfied (e.g., minimum recall), and the optimization goal (e.g., query time or memory usage). The recommender establishes a set of meta-instances simulating different parametrizations for the input dataset and infers the corresponding performance measures (meta-targets). Finally, it ranks the meta-instances according to the performance measure and optimization goal and returns the top parametrization to the user. Notice that even though the performance measurement is hardware-dependent, an induced meta-model using meta-targets from this performance measurement should be effective for other hardware as our recommender is based on relative performances using ranking.

### 3.1 Overview of a Recommender Instantiation Using Random-Forests

This section details the main aspects of an instantiation of the proposed recommender for graph-based methods to index image datasets. For this work, we selected the graph-based methods *Brute-kNNG*, *NN-Descent*, and *NSW* because of their importance as base methods for approximate similarity search. We considered *k-NN* queries using the *GNNS* algorithm because it is efficient and flexible to improve the recall besides being applicable to all types of graphs selected. These types of graphs have the construction parameter *NN* in common, which refers to the number of neighbors each element is connected to in the graph. The *NN-Descent* and the *NSW* also have specific construction parameters whose impact is not as important to the methods' performance as the impact of the *NN* parameter [24]. Thus, we arbitrarily fixed these parameters values, being $\rho = 0.5$ for the *NN-Descent*, and $efConstruction = 100$ for the *NSW*. Therefore, given the user input, our proposal recommends the graph type, the construction parameter *NN*, and the query parameter *R*.

**Input Datasets.** We have employed real and synthetic datasets to analyze the behavior of each graph-based method for different configurations. The real datasets contain features from images: *Color Moments*, *Texture*, and *Color Histogram* are feature vectors extracted from 68,040 photos obtained from Corel, with dimensionalities 9, 16 and 32, respectively; *MNIST*, the pixels of a collection of 70,000 images of handwritten digits comprising 784 dimensions; and *ANN-SIFT1M*, which is a collection of 1,000,000 SIFT features (128 dimensions). The

63 synthetic datasets employed were generated following a Gaussian distribution, varying the size, the dimensionality, the number of clusters, and the distribution standard deviation, using the Python library Scikit-learn[2].

**Meta-dataset.** To build our meta-dataset we first performed the characterization of each dataset. Most of the meta-features employed were based on general (*cardinality, dimensionality*), statistical (*sd, skewness, t_mean, var, nr_norm, nr_outliers, median, min, range, iq_range, kurtosis, mad, max, mean*), and information-theoretical (*attr_ent, inst_to_attr*) measures[3]. We also included the Intrinsic Dimensionality (ID) of the datasets as it has often been employed in the field of similarity search to measure a dataset complexity [4]. We used the Maximum Likelihood Estimation [15] to estimate the ID, and the tool PyMFE [23] to extract the remaining measures from the datasets. Finally, the graph type and its configuration were also employed as meta-features.

The average query time and recall obtained by each configuration of each graph-based method were used as meta-targets in the meta-dataset. The performance measurement was performed using implementations in the C++ library NMSLib (Non-Metric Space Library) [6]. The queries employ the Euclidean distance ($L_2$). We used a superset of the results of the experiments carried on a previous work, which includes executions for combinations of the parameters $NN \in \{5, 10, 25, 40, 55, 70, 100, 130, 150\}$ and $R \in \{1, 5, 10, 20, 40, 80, 120, 160, 200, 240\}$. For additional details on the experiment settings, refer to [24].

**Meta-model.** We used the implementation for Random Forest on Scikit-learn (with the following parameters: $n_estimators = 100$, $criterion =$ "*mse*", and $min_samples_split = 2$), to induce the meta-models, and a 5-fold Cross-Validation strategy to validate them. We selected the RF for its great prediction performance, reported in several recent works [9,11], simple parameterization [7], and capacity to evaluate feature importance. The recommender is composed by two Random Forests; one induced to predict the recall of a *k-NN* query using a specific graph configuration over a dataset, and the other to predict the query time. The recall is employed to filter the configurations that satisfy the user constraint about the minimum acceptable result quality while the query time is used for ranking the configurations according to the provided optimization (memory or query time).

## 4 Experimental Results

### 4.1 Analysis of Meta-feature Importance

Here we present an analysis of the importance rate of each meta-feature to predict the meta-target. These rates were measured by the meta-models performing a 5-fold cross validation over the meta-dataset. Results are presented in

[2] https://scikit-learn.org/.

[3] A detailed description of the general, statistical and information-theoretical meta-features is available in https://pymfe.readthedocs.io/en/latest/api.html.

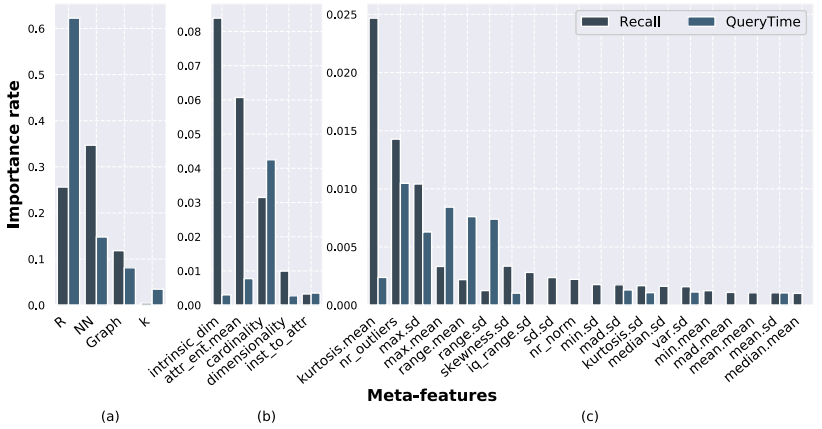



**Fig. 4.** The importance rate of each meta-feature per meta-target and category: (a) graph configurations, (b) general and info-theoretical, and (c) statistical.

Fig. 4. Overall, for both meta-targets evaluated, the most relevant meta-features were the construction parameter $NN$, the search parameters $R$ and $k$ and, the graph type. Nonetheless, other features also contribute to the prediction. In the meta-models, for each tree split from a set of descriptors, all the following splits depend on the graph type and its parameters, as these are the meta-features that refine and determine the final behavior of the proximity graph over a given set of dataset descriptors. Regarding recall, we can observe that the most relevant meta-feature is the construction parameter $NN$. This is implied by the fact that the more edges a graph has, the better its query recall rate is. Similarly, for query time, we have the query parameter $R$ as the most important meta-feature as the higher the number of restarts $R$ is, the longer the query execution time is, and vice-versa. Moreover, the high importance rate of the ID measure for recall analysis is an interesting result. Excluding the graph configuration meta-features, the ID had the highest rate, thus, the embedding dimensionality showed no relevance in this case.

### 4.2 Prediction Accuracy of the Meta-models

Subsequently, we present our results on the prediction accuracy of the meta-models regarding the real datasets used in this work. The synthetic datasets were added to the meta-database to provide a wider diversity of dataset characteristics. We evaluated our approach by using three different strategies: i) generic meta-model ($GMM$) – all meta-instances of our meta-dataset regarding all datasets were used for meta-training, except for the meta-instances regarding the goal dataset, which was used for meta-testing; ii) tuned meta-model using grid search ($TMM$-$GS$) – all meta-instances of our meta-dataset regarding all datasets (except for the ones regarding the goal dataset) plus meta-instances generated by the grid search performed over the goal dataset were used for

**Table 1.** Relative performances of the generic and tuned meta-models.

| Goal Dataset | GMM | | | | TMM-GS | | | | TMM-S | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Recall | | Query Time | | Recall | | Query Time | | Recall | | Query Time | |
| | $r^2$ | RMSE | $r^2$ | RMSE | $r^2$ | RMSE | $r^2$ | RMSE | $r^2$ | RMSE | $r^2$ | RMSE |
| Histogram | 0.350 | 0.135 | 0.980 | 0.249 | 0.605 | 0.130 | 0.961 | 0.338 | 0.996 | 0.012 | 0.998 | 0.068 |
| MNIST | 0.765 | 0.111 | 0.694 | 1.097 | 0.617 | 0.173 | 0.920 | 0.559 | 0.997 | 0.014 | 0.998 | 0.068 |
| Moments | 0.955 | 0.034 | 0.989 | 0.179 | 0.973 | 0.031 | 0.979 | 0.241 | 0.991 | 0.019 | 0.998 | 0.065 |
| SIFT | 0.807 | 0.132 | 0.932 | 0.524 | 0.568 | 0.247 | 0.803 | 0.932 | 0.983 | 0.049 | 0.984 | 0.260 |
| Texture | 0.978 | 0.024 | 0.962 | 0.344 | 0.990 | 0.022 | 0.951 | 0.378 | 0.996 | 0.012 | 0.998 | 0.058 |

meta-training and the goal dataset was used for testing; and iii) tuned meta-model using subsets (*TMM-S*) – meta-instances of our meta-dataset regarding all synthetic datasets (except for the ones regarding the goal dataset) plus meta-instances of subsets of the real datasets were used for meta-training, the remaining meta-instances were used for meta-testing. The first strategy simulates generating a recommendation for an unseen input dataset; the second one simulates a fine tuning of the meta-models by increasing the meta-dataset with meta-instances generated by a grid search with a limited parameter space; and lastly, the third one simulates a scenario in which the meta-model already knows datasets with similar properties to the input dataset.

Table 1 presents the relative performance achieved by each induced meta-model considering two evaluation metrics: the Coefficient of Determination ($r^2$) and the Root Mean Squared Error (*RMSE*). Both measure the predicted values by meta-models against true values (reached by the graph-based methods). Good fits for these metrics are, respectively, high and low values. From the results, we can observe that the strategy using the generic meta-models (GMM) reached good scores for query time and fair scores for recall for most of the datasets. This is because meta-features were more supplementary for query time than for recall. For TMM-GS, it was expected a small improvement compared to the GMM. Although in the most cases the performance was relatively similar, at final recommendation the TMM-GS outperformed GMM (further details in the next section). On the other hand, the most tuned meta-model TMM-S achieved high scores for both recall and query time. Therefore, by investing some effort to generate meta-instances of the goal dataset, the user can achieve a superior recommendation.

### 4.3 Effectiveness of the Recommendation

Lastly, we discuss the effectiveness of the recommendation provided by our approaches compared to a Grid Search (GS). We emulated a grid search using the subset of entries in the meta-dataset such that $NN = \{1, 25, 70, 150\}$ and $R = \{1, 10, 40, 120\}$. These were the same parameters used to evaluate the *TMM-GS*. In this analysis, we set the constraint of achieving a minimum average recall of 0.90 and evaluated two optimization criteria: (a) the shortest query time, and

(b) the lowest memory usage. Figure 5 presents the recommendations provided by the differents strategies according to each criteria. The figures (a)–(c), refer to memory optimization, and the figures (d)–(f) refer to query time optimization.

Figure 5(a) and (d) show recommendations for subsets of *Color Histogram*, and Fig. 5(b) and (e) of *Color Moments*. For memory optimization, the GMM overcame the GS in all cases, reaching the optimal most times. For query time optimization, the GMM performed worse than GS around $\frac{1}{4}$ of the times, where one of them was a wrong recommendation. We consider a wrong recommendation when the method fails to satisfy the recall constraint due to a poor prediction. In these cases, the *NN* or the query time provided by the meta-models may be smaller than the optimal values (e.g., Fig. 5(e) for $8k$ instances).

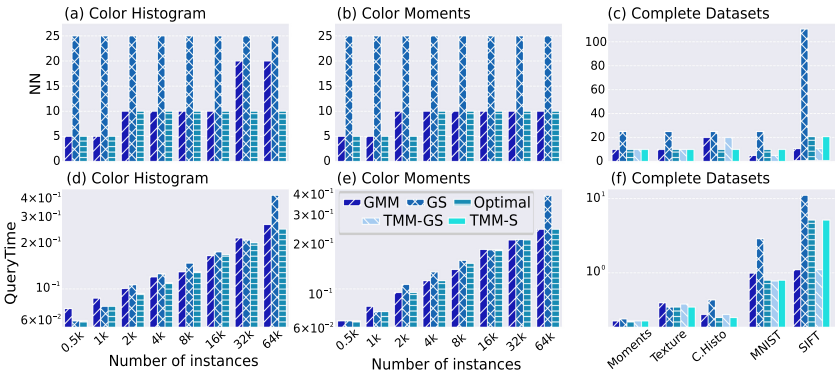



**Fig. 5.** Comparison of the recommendations provided by the methods.

Figure 5(c) and Fig. 5(f) show recommendations for the complete datasets provided by all the methods, including the meta-models TMM-GS and TMM-S. Overall, the TMM-GS outperformed the GMM, and the TMM-S was the best strategy, consistently reaching the optimal. For the datasets *Color Moments*, *Texture* and *Color Histogram*, our meta-models were more effective than the GS. However, for the datasets *MNIST* and *SIFT*, which are the most complex datasets used in this work due to their high dimensionality, the GMM and the TMM-GS provided wrong recommendations while the TMM-S achieved optimal results.

To better understand how much the strategies provide wrong predictions, Fig. 6 shows the true recall rates with their corresponding predictions split into intervals. In Fig. 6(a), we can observe that the GMM is able to provide predictions very close to the true values for most of the tested cases, however, it falls short for *SIFT* (Fig. 6(b)). Nevertheless, Fig. 6(c) presents the performance reached by the TMM-S where there was a huge improvement. Such a behavior not only reinforces the need of continuously enhancing the meta-dataset to improve the model generalization, but also highlights the power of our tuning proposal.

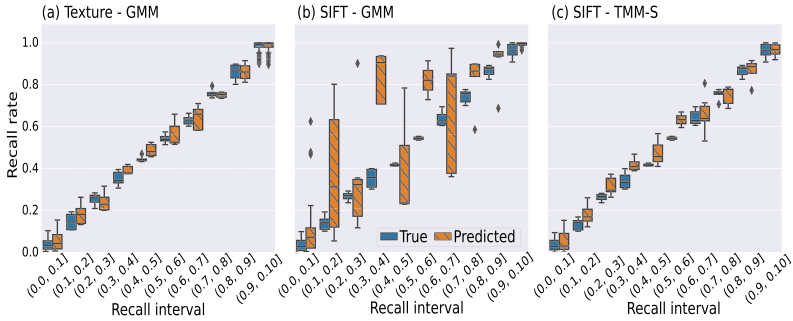



**Fig. 6.** Accuracy of the predictions of the meta-models per recall interval.

## 5 Conclusion and Further Research

In this paper, we proposed an intelligent approach capable of recommending a graph-based method and its configurations to perform similarity searches. The main idea consists in using meta-learning techniques to estimate the relative performance of different graph configurations for the given dataset to select the most suitable one. We presented an Instantiation of our approach for image databases using Random Forests. We also evaluated three variations of this instantiation, GMM, TMM-GS and TMM-S, and compared them to a standard grid search.

Our results showed that, for many situations, the generic approach GMM tied or outperformed the grid search, without requiring the user to execute performance measures on the goal dataset. However, it failed to provide valid recommendations for the most complex datasets tested. The tuned approach TMM-GS improved over GMM, nevertheless, it still provided a few invalid recommendations. On the other hand, the approach tuned with subsets, TMM-S, approached the optimal results, but it is an expensive approach though.

For future works, we intend to generate other instances of our approach with richer meta-databases, exploring more meta-features to describe the input datasets and including other meta-targets, such as construction time and real memory usage to store the structure.

## References

1. Aguiar, G.J., Mantovani, R.G., Mastelini, S.M., de Carvalho, A.C.P.L.F., Campos, G.F.C., Junior, S.B.: A meta-learning approach for selecting image segmentation algorithm. Pattern Recognit. Lett. **128**, 480–487 (2019)
2. Amato, G., Gennaro, C., Savino, P.: MI-File: using inverted files for scalable approximate similarity search. Multimedia Tools Appl. **71**(3), 1333–1362 (2014)
3. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: ANN-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. In: Beecks, C., Borutta, F., Kröger, P., Seidl, T. (eds.) SISAP, pp. 34–49. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68474-1_3

4. Aumüller, M., Ceccarello, M.: Benchmarking nearest neighbor search: influence of local intrinsic dimensionality and result diversity in real-world datasets. In: EDML SDM. CEUR Workshop Proceedings, vol. 2436, pp. 14–23 (2019). CEUR-WS.org
5. Baranchuk, D., Babenko, A.: Towards similarity graphs constructed by deep reinforcement learning. CoRR abs/1911.12122 (2019)
6. Boytsov, L., Naidan, B.: Engineering Efficient and effective non-metric space library. In: Brisaboa, N., Pedreira, O., Zezula, P. (eds.) SISAP 2013. LNCS, vol. 8199, pp. 280–293. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41062-8_28
7. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
8. Dong, W., Charikar, M., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: WWW, pp. 577–586. ACM (2011)
9. Eggensperger, K., Lindauer, M., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Efficient benchmarking of algorithm configurators via model-based surrogates. Mach. Learn. **107**(1), 15–41 (2017). https://doi.org/10.1007/s10994-017-5683-z
10. Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., Zhang, H.: Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: Walsh, T. (ed.) IJCAI, pp. 1312–1317. IJCAI/AAAI (2011)
11. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: methods & evaluation. Artif. Intell. **206**, 79–111 (2014)
12. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Vitter, J.S. (ed.) STOC, pp. 604–613. ACM (1998)
13. Traina, C., Traina, A., Seeger, B., Faloutsos, C.: Slim-trees: high performance metric trees minimizing overlap between nodes. In: Zaniolo, C., Lockemann, P.C., Scholl, M.H., Grust, T. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 51–65. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46439-5_4
14. Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N.: The case for learned index structures. In: Das, G., Jermaine, C.M., Bernstein, P.A. (eds.) SIGMOD, pp. 489–504. ACM (2018)
15. Levina, E., Bickel, P.J.: Maximum likelihood estimation of intrinsic dimension. In: NIPS, pp. 777–784 (2004)
16. Li, W., Zhang, Y., Sun, Y., Wang, W., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0). CoRR abs/1610.02455 (2016)
17. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Inf. Syst. **45**, 61–68 (2014)
18. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISAPP, pp. 331–340. INSTICC Press (2009)
19. Naidan, B., Boytsov, L., Nyberg, E.: Permutation search methods are efficient, yet faster search is possible. PVLDB **8**(12), 1618–1629 (2015)
20. Navarro, G.: Searching in metric spaces by spatial approximation. VLDB J. **11**(1), 28–46 (2002)
21. Paredes, R., Chávez, E.: Using the $k$-nearest neighbor graph for proximity searching in metric spaces. In: Consens, M., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 127–138. Springer, Heidelberg (2005). https://doi.org/10.1007/11575832_14
22. Paredes, R., Chávez, E., Figueroa, K., Navarro, G.: Practical construction of $k$-nearest neighbor graphs in metric spaces. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 85–97. Springer, Heidelberg (2006). https://doi.org/10.1007/11764298_8

23. Rivolli, A., Garcia, L.P.F., Soares, C., Vanschoren, J., de Carvalho, A.C.P.L.F.: Towards reproducible empirical research in meta-learning. CoRR abs/1808.10406 (2018)
24. Shimomura, L.C., Oyamada, R.S., Vieira, M.R., Kaster, D.S.: A survey on graph-based methods for similarity searches in metric spaces. Inf. Syst. 101507 (2020)
25. Smith-Miles, K.A.: Towards insightful algorithm selection for optimisation using meta-learning concepts. In: IJCNN, pp. 4118–4124. IEEE (2008)
26. Vanschoren, J.: Meta-learning: a survey. CoRR abs/1810.03548 (2018)