Class-Constraint Similarity Queries*

Jessica A. de Souza
Institute of Mathematics and Computer Sciences
University of Sao Paulo
Sao Carlos, SP, Brazil
jessicasouza@usp.br

Agma J. M. Traina
Institute of Mathematics and Computer Sciences
University of Sao Paulo
Sao Carlos, SP, Brazil
agma@icmc.usp.br

Sebastian Michel
Department of Computer Science
TU Kaiserslautern
Kaiserslautern, Germany
smichel@cs.uni-kl.de

ABSTRACT

Similarity searching is a widely applied concept on multimedia or complex data, such as images, videos, time-series, among others. Therefore, it is important to look at the execution of specific query types, e.g., constrained k-nearest neighbor that is directly based on bounded regions. In this paper, we present the Class-Constraint k-Nearest Neighbor (*CCkNN*) query, which goes beyond the traditional constrained k-nearest neighbor, because our *CCkNN* works for any specific categories of data points. The proposed *CCkNN* aims at accelerating the process of class-constraint similarity query execution by taking advantage of performing queries on multiple metric access methods regarding the class dimensions of the objects of each index. Additionally, this strategy identifies which index is more appropriate to run class-constraint on the k-nearest neighbor queries. Experimental results based on several datasets, including synthetic and real ones, show that our strategy can reduce the number of distance calculations in up to two orders of magnitude while keeping a high-quality retrieval, according to the classes of the objects queried.

CCS CONCEPTS

• **Information systems** → *Data management systems*; **Data structures**; **Data access methods**; *Proximity search*;

KEYWORDS

Metric access methods; inverted index; complex datasets; k-nearest neighbor

ACM Reference Format:
Jessica A. de Souza, Agma J. M. Traina, and Sebastian Michel. 2018. Class-Constraint Similarity Queries. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167192

1 INTRODUCTION

The classical k-nearest neighbor (k-NN) query returns the k objects with the smallest distances to a query object. In this paper, we are interested in efficiently and effectively answer queries that impose additional class constraints on the k-NN retrieval operation, in order to enhance the execution algorithm and get the results restricted to the desired classes. Combining k-NN retrieval with other constraints is more challenging than the combination of the traditional constraints, based only on identity or order relationships. As an example, the traditional constraints are commutative, that is, if two constraints need to be applied, they can be executed in any order or even evaluated at once, as a comparison conjunction. However, when a query involves a k-NN and any other constraint, the execution order changes the result. As an example, retrieving the five mammographies that are the most similar to a given one, and then selecting those that have a calcification, may return less than five results, whereas selecting those with calcification and from then retrieving the five most similar always return five mammographies.

As retrieving data based on k-NN constraint is even more time-consuming than the retrieval based on traditional constraints, employing index structures is even more important than the usual. Thus, the common strategy is to create an index structure on the attribute that should be retrieved by similarity [11]. Moreover, to get the largest benefit of using such index, the first constraint to be executed must be the similarity one. Thus, previous study on combining k-NN with other constraints often consider executing the k-NN first and then apply the other restrictions, even if the answer results in less than the desired number of objects. The alternative is to execute the k-NN retrieval based on the other constraints and then execute the also costly intersection of both intermediate results.

In this paper, we propose a new indexing scheme to answer the Class-Constraint k-Nearest Neighbor queries *CCkNN*. It is based on searching for similarity employing multiple indexes, what allows an efficient execution of the k-NN restriction, and at the same time taking into account a set of classification constraints Q_c. Our strategy retrieves the correct number of elements desired, so it is both efficient and effective to meet the user's demands. That is, our proposal is broadened than a constrained k-NN [3], which takes advantage of applying constraints regarding certain spatial conditions for accelerating the process querying.

So far, constrained queries have been exploited in both academia and industry. Examples of studies that have proposed strategies to address this issue can be found in [1, 7, 9]. In these studies, the authors investigated new strategies to improve the performance of similarity queries. For example, in [1] the authors employed k-nearest neighbor queries on road networks to search for the closest points of interest by their road network distance. In [7] the authors developed a new type of query, called the direction-constrained k-nearest neighbor to apply in geospatial systems. The study of [9]

*This research has been supported by FAPESP, CAPES and CNPq.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167192

applied user's preference query requirements on privacy protection in location-based services. However, in spite of the remarkable progress made in the last years, similarity queries still deserve attention, because of the resulting quality.

Motivated by the aforementioned reasons, this paper presents the following contributions:

(1) It provides a strategy to minimize the time required to answer class-constraint similarity queries. The time spent to run a query is a basic measure when evaluating an index structure. Considering Metric Access Methods, time is also directly related to the computation of distances;
(2) It employs queries on multiple indexes in-memory, considering class constraints;
(3) It shows how to combine the class dimensions and to employ them on the traditional k-NN query;
(4) Finally, it presents experimental results comparing the performance of executing the *CCkNN* strategy on multiple metric access methods *vs* on the Inverted Index.

The rest of the paper is organized as follows. Section 2 describes the reasons for the investigation presented herein. Section 3 reviews basic concepts and the proposed idea is presented with more details in Section 4. Section 5 describes the methodology and Sections 6 reports experiments involving synthetic and real datasets. The conclusions are summarized in Section 7.

2 PROBLEM STATEMENT

In this paper we are interested in exploring the following problem: given a set of complex data[1] $\mathcal{S} \in \mathbb{S}$ obtained from labeled datasets, find the k-nearest neighbours from a query object (s_q) regarding to its set of class constraints (Q_c, where $|Q_c| \leq |\mathfrak{C}|$) in a timely manner. The following definition is needed in this context.

Definition 1. *(Class dimensions) Given a set of class dimensions* $\mathfrak{C}$=$\{C_1, C_2, ..., C_n\}$*, each class dimension is a set* C_i*, and the cardinality for each class dimension* $C_i \in \mathfrak{C}$ *can be distinct, where* $1 \leq i \leq n = |\mathfrak{C}|$.

In Figure 1 we illustrate a k-NN query to $k = 3$, considering a sample of the classical iris dataset[2] with unidimensional classification "*Specie*", $|\mathfrak{C}| = 1$ where the *Specie*={Iris-setosa, Iris-versicolor, Iris-virginica}, and its cardinality is three. Thus, for $\mathfrak{C} = \{Specie\}$, the set of class dimension C_1 = {Iris-setosa, Iris-versicolor, Iris-virginica}. In summary, a class-constraint k-NN query can increase the number of distance computations to find the k objects according to a set of classes associated to s_q. Thereby, the strategy presented herein was guided by the following question: *"Which indexing method is the best choice to run class-constraint similarity queries?"* More specifically, as input we assume a set $\mathcal{S}$ of objects and a dissimilarity measure expressed as a metric distance function $d()$. Each object $s_i \in \mathcal{S}$ is further associated to a set of class dimensions where each object falls into, in such a way that the set $\mathcal{S}$ has an attribute for each classification dimension. For instance, consider a set composed of mammographic exams, indexed by its metric distance where each object is classified following two dimensions (2 class types): lesion and lesion-type. Both dimensions have cardinality 2, because lesion={calcification, mass} and lesion-type={benign, malignant}.

Each class is of the form <attribute:value>[3]. For example, class dimensions can be combined {<lesion:benign>, <lesion-type:mass>}, and in this case it corresponds to every object having a *mass lesion* evaluated as *benign*. For a class <a:v>, the attribute "a" is referred to as a class dimension. We assume that one object can be in only one class per class dimension, e.g., each object has one *lesion* and one *lesion-type*. We consider a query object s_q and a set of classes Q_c that is either associated to the query object, or it is specified at query time. That is, if s_q has a set of classes Q_c= {C_1={lesion:malignant}, C_2={lesion-type:calcification}}, a proper index would be one, which only keeps all the objects $\in$ C_1={lesion:malignant} and C_2={lesion-type:calcification}. However, if Q_c = {C_1={lesion:malignant}}, in this case, a proper index would be one that only keeps all the objects $\in$ C_1={lesion:malignant}. Thus, the goal is to compute the k-NN objects from the s_q object with the class constraints that each query needs to capture for all the classes of Q_c, and also getting the results in a timely manner.

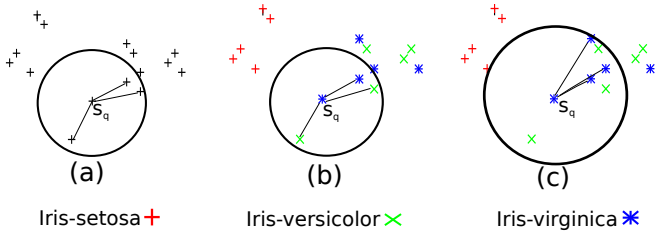


Figure 1: Illustration of a class-constraint to k-NN query (right) considering $k = 3$ regarding the s_q object and Q_c={C_1 = {Specie:Iris-virginica}}. (a) Classical k-NN and unlabeled data, (b) Classical k-NN and labeled data and (c) After applying the class-constraint k-NN on labeled data.

Note that in this example, two of the 3-nearest objects of the Iris-virginica class are far from the query center than other objects from the class Iris-versicolor. However, the results are the ones asked by the query.

3 BASIC CONCEPTS

This section presents the fundamental concepts supporting this study, such as a brief description of the main aspects related to the metric access methods. Table 1 summarizes the symbols used throughout this paper.

3.1 Index Structure

The main goal of an index structure is to accelerate the data access, mostly by minimizing the number of distance computations and/or disk accesses [8]. For processing similarity queries, the literature has presented several metric access methods (MAMs). These methods are specialized data structures based on trees, which aim at reducing the time regarding the processing of a query. For example, the pioneer dynamic metric access method, called M-tree [2], has guided

[1] that is, a document collection, images, videos or audios are complex data from which the features will be extracted from.
[2] http://archive.ics.uci.edu/ml/datasets/Iris?ref=datanews.io
[3] This notation is very generic and is reflected in JSON syntax

Table 1: Summary of symbols and definitions

Symbols	Definitions
$\mathcal{S}$	Set of objects in domain $\mathbb{S}$
$\mathbb{S}$	Domain of objects
$\mathcal{R}$	Answer set of a similarity query
k	The number of neighbors in a NN query
s_q	a Query object (or query center)
$d()$	Distance function
$\mathcal{Q}_c$	Set of class constraints to s_q
C_t	Set classes regarding to t_j in $\mathcal{T}$
$\mathcal{T}$	Answer set of labeled trees
t_j	Object of $\mathcal{T}$
$\mathfrak{C}$	Set of class dimensions of the dataset
C_i	Set of classes from a class dimension in $\mathfrak{C}$
n	Number of class dimensions in $\mathfrak{C}$
s_i	Object of $\mathcal{S}$

the development of other methods and several variations [8, 11, 14], such as the Slim-tree [13]. Therefore, it is important to note that our objective is to employ only MAMs, which are structures for both multidimensional and non-dimensional data, unlike R-tree and its descendants, which apply to multidimensional data [12].

3.2 Similarity Search

A classical similarity query processing aims at retrieving objects from the dataset that are similar to a given reference object. For this intent, the literature presents two main types of queries that compare a reference object with those stored in the dataset [14]: *Range query* (Rq) and k-Nearest Neighbor (k-NN query). However, this traditional strategy does not apply constraints on the similarity query. In this way, the demand for efficient techniques that require similarity computations (e.g., k-nearest neighbor queries) has motived the development of new strategies. For example, the study in [3] presents a constrained k-nearest neighbor query returning the k closest objects within a region (r) specified by some constraints (i.e., one or multiple bounded regions to perform the queries). This type of strategy is directed to the queries in a bounded region by certain spatial conditions, rather than processing k-NN queries in the entire dataset (data space). Nevertheless, herein, we want to exploit the indexing of objects with classes (labeled datasets) and apply their classes on a k-NN query. In other words, class containment is not bounded to a region of the metric space. This feature guided the selection of the baseline strategies, Inverted Index and Single Metric Access Method algorithms are the baseline algorithms used to help executing k-NN queries. The next section describes our proposed method: the CCkNN strategy.

4 PROPOSED IDEA: THE CCkNN

CCkNN is a new strategy for finding the k-nearest neighbors from a query object s_q according to a set of constraints $\mathcal{Q}_c$. In general, the algorithm presented herein aims at optimizing similarity comparisons by building multiple indexes and aiding to ensure data quality. That is, the proposal takes into account not only spatial constraints, but also any type of class constraints. Figure 1 illustrates the general problem tackled by our research, i.e., the impact of applying class constraints on the search space that needs to be investigated to answer a k-NN query. We assume that an index is created on the entire dataset, what was sketched in Section 2. The query, which is shown there, asks for $k = 3$ nearest neighbors from s_q object regarding the $\mathcal{Q}_c$={C_1 = {Iris-plant:Iris-virginica}}. It is important to note that, this dataset presents $|\mathfrak{C}|$=1 class dimension and the cardinality of the class dimension $|C_1|$=3.

Thus, for example, if 99% of the objects are of class Iris-virginica, in expectation only $(1/0.99)\times k \approx 1.01\times k$ nearest objects need to be retrieved to find the k-nearest neighbors of $\mathcal{Q}_c$={C_1={Iris-plant:Iris-virginica}}. If, in contrast, only 1% of the objects are *Iris-virginica*, on average $100\times k$-nearest neighbor need to be retrieved/computed and checked for class membership. This exemplifies the inherent cost variability to process a constraint query using the traditional strategy.

Now, assume that we only got $|\mathfrak{C}|$ class dimensions from the dataset composed of mammographic exams (see Section 2), which has $|\mathfrak{C}|$=2 class dimensions and each class dimension with $|C_i|$=2 classes. Consequently, the set of class dimensions can be drawn as $\mathfrak{C}$ = {C_1={lesion:benign, lesion:malignant}, C_2={lesion-type:mass, lesion-type:calcification}}. The obvious idea would be to create one index for each class. If each object can take on only one of the classes (which is the case presented in Figure 1), then this indexing does not create any storage overhead. But then, if $|\mathcal{Q}_c|$=1 class dimension, only one index needs to be accessed. In general, $|\mathcal{Q}_c| > 1$ or $|\mathcal{Q}_c| = |\mathfrak{C}|$, which is the total number of class dimensions of the dataset.

Thus, to make things more interesting, we assume that each object can only be in one class for each class dimension. Let us consider the question: *how many indices would we need to create?* Consider $|\mathfrak{C}|$ class dimensions with $|C_i|$ classes each dimension. Let $|\mathcal{T}|$ to represent the cardinality of the set of indexes built according to the number of class combinations. This measure can be calculated as shown in Equation 1. For example, for the set of classes of mammographic exams (presented in Figure 2) for $|\mathfrak{C}|$=2, we have $|C_1|$=2 and $|C_2|$=2 for each class dimension, thus $|\mathcal{T}| = [(2+1).(2+1)] = 9-1$, or 8 indexes. It is important to note that the term +1 for each dimension corresponds to the combination of set { } and the term −1 comes because we cannot build one index without objects. That is, we consider all the distinct combinations from $\mathfrak{C}$.

$$1 \leq |\mathcal{T}| \leq \left[\prod_{i=1}^{|\mathfrak{C}|} |C_i| + 1\right] - 1 \tag{1}$$

From this context, *how large are the individual indexes?* If we have $|\mathcal{S}| = 100$ objects in total, and assuming uniform distribution, each class contains $\frac{|\mathcal{S}|}{|C_i|}$, where the cardinality for each class dimension $|C_i| = 2$. A (conjunctive) combination of two class dimensions contains $\frac{|\mathcal{S}|}{|C_i|^2} = 25$ objects for each class. In general for $|\mathfrak{C}|$ class dimensions together, $\frac{|\mathcal{S}|}{|C_i|^{|\mathfrak{C}|}}$ objects fall into it. Thereby, *how redundant are these classes?*, consider a specific object consisting of $|\mathfrak{C}| = 2$ class dimensions, it is clear that all objects that fall into the classes are also contained in the classes of the $|\mathfrak{C}| - 1$ class dimensions. Thus, one object s_i will be indexed on all the

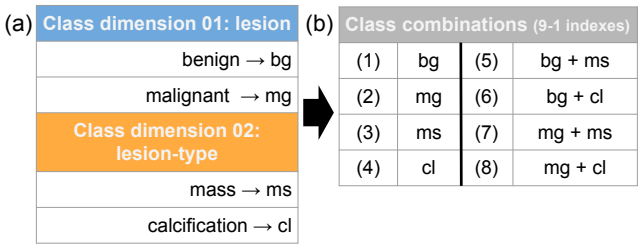


Figure 2: Illustration of the classes of diagnoses of mammographic exams. (a) $|\mathfrak{C}| = 2$ class dimensions and each type has $|C_i| = 2$ classes, (b) all the class combinations without overlapping of class of the same dimension and for each combination one index will be built.

labeled indexes according to its class set, e.g., an object of $\mathfrak{C} = \{C_1 = \{\text{lesion:benign}\}, C_2 = \{\text{type-lesion:calcification}\}\}$ will be indexed on the labeled indexes as $t_{lesion:benign}$, $t_{type\_lesion:calcification}$ and $t_{lesion:benign,type\_lesion:calcification}$. For class dimensions *lesion* and *lesion-type* each one with 2 classes, we have already $(9-1)$ indexes to create (see Figure 2 (b)).

Algorithm 1 outlines the *CCkNN* strategy. It is important to highlight that, in line 1 the parameter Q_c influences the choice of the indexing structures to execute the queries. Overall, the resulting query exhibits the objects nearest to s_q regarding the set of class constraints Q_c. Our strategy, seeks a proper tree $\in \mathcal{T}$ to execute class-constraint k-NN queries. Thus, *CCkNN* can avoid queries on a high number of labeled trees. The loop (in lines 5–8) searches for a labeled tree with the same dimensions in Q_c.

Algorithm 1: CCkNN

```
1  input: k, s_q, Q_c, T;
2  output: the k nearest neighbors from s_q object according to
     its set of class constraints Q_c;
3  begin
4  R ← {};
5  foreach t ∈ T do
6      if Q_c ⊂ C_t then
7          find the k-nearest neighbors from s_q;
8          R ← k;
9  if R ≠ ∅ then
10     Return the k-nearest neighbors;
11 end
```

5 METHODOLOGY

In order to assess the proper indexing strategy to process class-constraint queries, we measured the number of distance calculations to build the structures and to execute the queries. Besides that, the required memory size is estimated in megabytes to analyze each indexing structure built, regarding all the class combinations. Inverted Index and a metric access method representative were applied. We chose the Slim-Tree MAM [13] because it is well known in the literature to generally perform well. These indexing strategies can be considered representative schemes of processing similarity queries:

Inverted-Index employs buckets that store objects formed by the combination of a class and a set of feature vectors, what allows fast retrieval of individual objects based on their classes.

Slim-Tree is a dynamic and balanced tree data structure, constructed from the leaves toward the root (bottom-up). As other MAMs (e.g., M-Tree [14]), the Slim-Tree gathers the elements of a dataset in fixed-size pages, such that each page corresponds to a node of the tree. The experiments were performed on a k-nearest neighbor query algorithm, which performs a k-nearest neighbor query using a global priority queue based on a dynamic heap to improve its performance [6].

Multiple indexes for class combinations according to the discussion presented in Section 4. For each combination of class dimensions, one metric index will be built and for each set of combinations one Inverted Index will be built to keep the combinations on the buckets. For instance, given a set of classes with $|\mathfrak{C}| = 4$ class dimensions and $|C_i| = 3$ classes per class dimension, the number of metric indexes will be 255. However, the number of inverted indexes (maps) is the number of class dimensions, in this case, it is equal to 4, so only four inverted indexes will be built. It is important to note that, the class combinations were built by a recursive algorithm.

6 EXPERIMENTAL EVALUATION

This section reports the experimental analysis regarding the average running time to execute the queries according to two indexing strategies: Inverted Index and Metric Access Method. The results show the average of 50 k-nearest neighbor queries for each value of k using distinct sets of randomly selected query centers. All experiments were performed on an Intel® Xeon® CPU X5650, 32GB DDR3 1333MHz RAM and Ubuntu® 16.04.2 (64-bit) GNU/ Linux OS and every strategy was implemented in C++.

6.1 Dataset Description

We employed synthetic datasets with a different number of classes, in order to enable a comprehensive evaluation of our strategy. Real datasets were also used to analyze the behavior of the algorithms in a real-world situation. The description of each dataset is presented in Table 2 along with its name, the total number of objects (#Objs.), the dimensionality of the dataset D, the number of class dimensions $|\mathfrak{C}|$, the number of classes for each set of class dimension $|C_i|$, and the distance function used $d()$. We employed every C_i to have the same cardinality to allow a direct comparison among datasets. It is important to note that different distance functions were employed in the experiments, with the intent to evaluate the adequacy of the proposed strategy in several situations.

6.2 Evaluating Multiple Indexes vs Single Indexes

The first set of experiments was carried out in order to compare the set of class constraints Q_c considering the same number of class dimensions of the dataset, that is, $|Q_c| = |\mathfrak{C}|$. The *efficiency* was measured by the number of distance calculations and by the

Table 2: Description of the synthetic and real-world datasets used in the experiments

Name	#Objs	D	$\|\mathfrak{C}\|$	$\|C_i\|$	$d()$	Description
uRand500_16d_3ct_2c	500,000	16	3	2	L_2	Synthetic vector data with Uniform distribution and $\sigma^2 = 0.1$. The process to generate the synthetic datasets is described in [2].
uRand500_16d_3ct_3c	500,000	16	3	3	L_2	
uRand500_32d_3ct_2c	500,000	32	3	2	L_2	
gRand500_32d_3ct_3c	500,000	32	3	3	L_2	Synthetic vector data with Gaussian distribution and $\sigma^2 = 0.01$. The dataset names also indicate the dimension, number of class dimensions and classes.
gRand500_32d_4ct_2c	500,000	32	4	2	L_2	
gRand500_32d_4ct_3c	500,000	32	4	3	L_2	
mammoHaralick_24d_3ct_2c	3,333	24	3	2	Canberra	Real datasets are categorized into four categories based on the view of the breast image, details in [5].
mammoZernike_36d_3ct_2c	3,333	36	3	2	$\tilde{\chi}^2$	

total time spent to compute distance-based queries. Figure 3 (a–f) shows the efficiency comparison of *Single-MAM* (queries on a single metric access method Slim-tree), *Single-InvIndex* (queries on a single Inverted Index), *CCkNN* (queries on a specific metric access method built according to the combinations of class dimensions) and *Multiple InvIndex* (queries on a specific Inverted Index built according to the set of combinations of class dimensions) for the synthetic datasets.

The *CCkNN* strategy outperformed every other for the synthetic datasets. Notice that Figure 3 shows the number of distance calculations in log scale. When compared to *Single-MAM*, if we observe the results obtained considering k=50, this strategy required up to 96, 05% fewer distance computations for the synthetic datasets with Uniform distribution (Figure 3 (a–c)) and up to 80 times fewer distance computations for the synthetic datasets with Gaussian distribution (Figure 3 (d–f)). Therefore, the time spent to execute the queries was also the lowest one (see Figure 4).

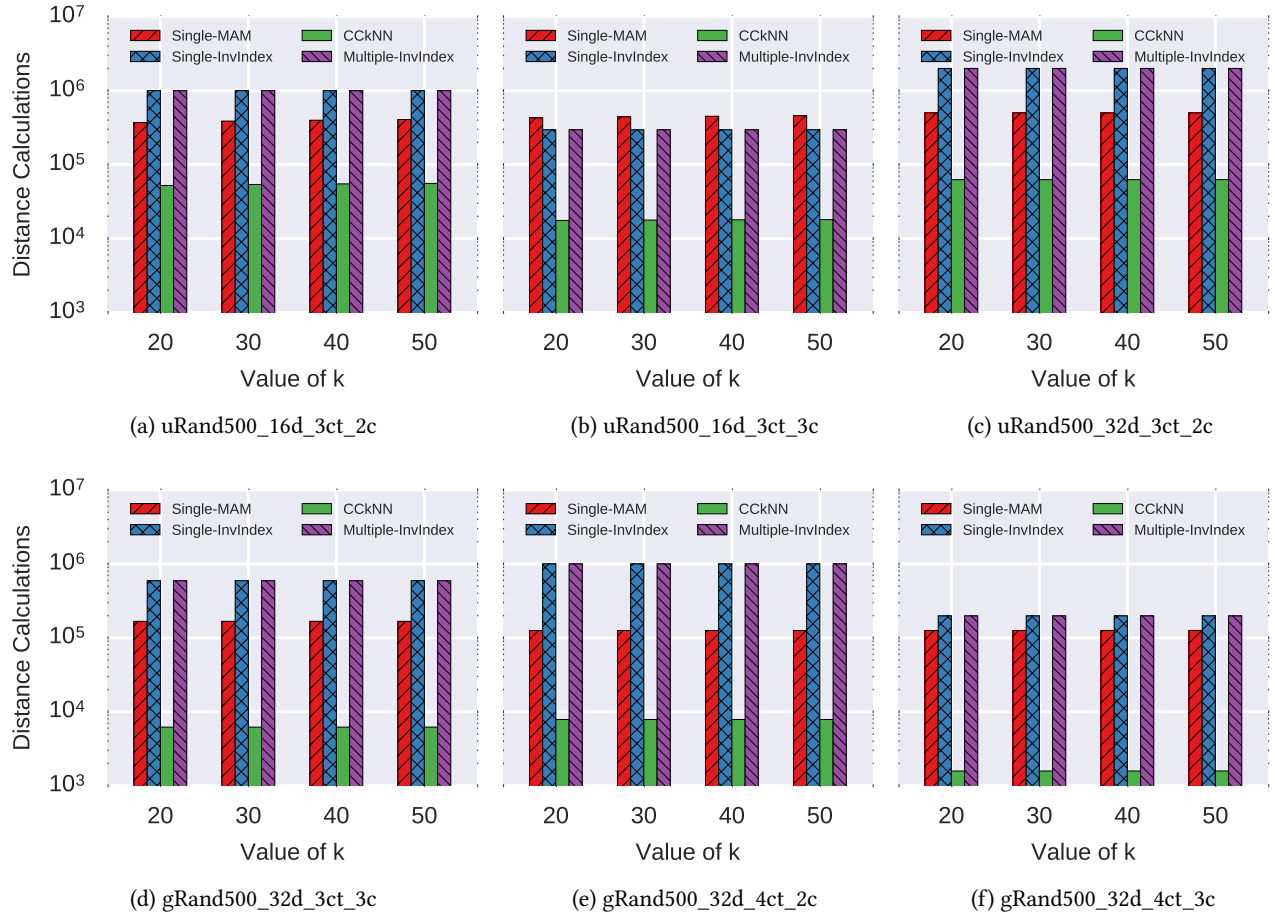
Particularly with respect to the behavior of the *Single-MAM* when compared to the *Single-InvIndex* and *Multiple-InvIndex* strategies, it is important to note that as the amount of both data and dimensions of the dataset increases, this strategy required a number of distance calculations with the same order of magnitude. However, due to the application of an incremental nearest-neighbor query algorithm, the experiments showed that this stratey spent more time to execute the queries (Figure 4 (a–c)) than the other strategies. Besides that, the experiments showed that keeping all objects on a metric access method to execute class-constraint k-NN queries can be costlier than to execute a scan on all objects of a bucket, which stores specific objects according to their classes. It has also been noted that, when $|Q_c| = |\mathfrak{C}|$, the time spent to execute the queries on a *Single-InvIndex* is the same when compared to a *Multiple-InvIndex* that stores all the class combinations for $|Q_c| = |\mathfrak{C}|$ (Figures 3 and 4). In general, the results related to the *Single-InvIndex* and *Multiple-InvIndex* strategies were worse than those achieved by the other strategies. This happens due to the sequential scan strategy that each one applies on their buckets to return the k nearest objects to the s_q query center. Consequently, these strategies required a higher number of distance computations.

6.3 Analyzing a Real Dataset

In this set of experiments we used the medical image datasets *mammoHaralick_24d_3ct_2c* and *mammoZernike_36d_3ct_2c*, in order to observe the behavior of our strategy in a real situation. In these experiments, the *Canberra* and $\tilde{\chi}^2$ distance functions were applied on the feature vectors. The feature extractors applied were a shape descriptor, given by *Zernike Moments* and a texture descriptor, given by *Haralick*. Yet, the number of class dimensions in Q_c does not varied. Figure 5 presents the results obtained for these datasets. Regarding the number of distance calculations (Figure 5 (a) and (b)), we can notice that, in general, the results related to the *CCkNN* presented the same behavior with both synthetic and real datasets. As expected, the trees built from fine-grained class combinations presented smaller number of objects indexed. Thus, when $|Q_c| = |\mathfrak{C}|$, *CCkNN* presented the smallest total query execution time.

When examining all the parameters evaluated in a conjunction, it is possible to conclude that the results obtained with the strategy presented herein led to a nice trade-off between the total time to execute the class-constraint k-nearest neighbor queries and the size of memory required to keep all the metric access methods in-memory. When compared to *Multiple-InvIndex*, observing the results obtained considering all the computed k for the *mammoZernike_36d_3ct_2c* and *mammoHaralick_24d_3ct_2c* datasets, the queries executed on a specific tree regarding its set of class outperformed the ones achieved by both *Single-MAM* and *Multiple-InvIndex*. Compared to *Multiple-InvIndex*, observing the results obtained considering that k equals to 50 for the *mammoZernike_36d_3ct_2c* dataset, the *CCkNN* strategy required 373 times less distance computations (see Figure 5 (b)). However, this strategy required more memory space to keep all the labeled metric indexes according to the class combinations. For example, for the *mammoZernike_36d_3ct_2c* dataset (see Figure 5 (f)), *CCkNN* required 6 times more memory than *Multiple-InvIndex*. Thus, another important remark is that, when the cardinality and dimensionality of the data, and the class dimensionality increase, our strategy will require more memory space, but on the other hand, computes significantly fewer distances. Thus, when the bottleneck is processing, because complex data may demand costly distance functions to be compared, the *CCkNN* is clearly the strategy of choice.

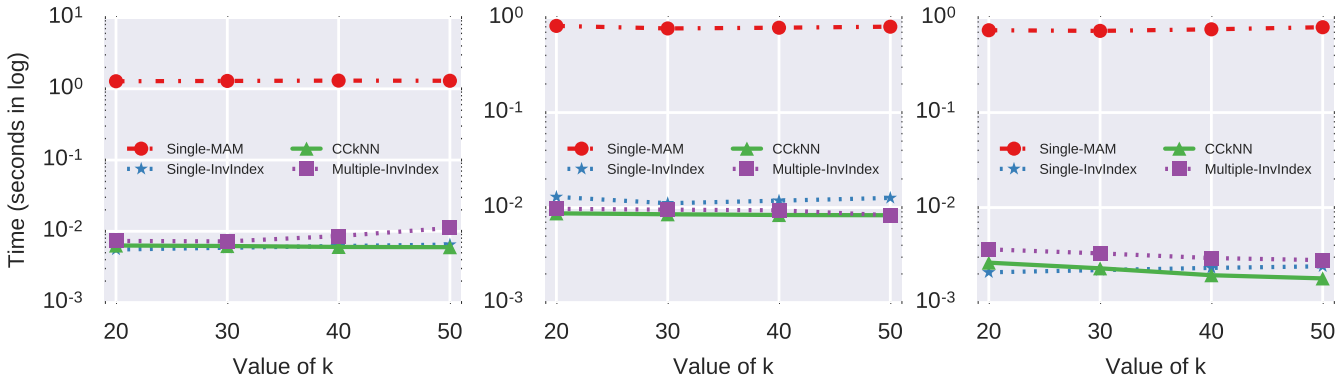
In summary, we can draw the following observations from the graphs showed in Figures 3, 4 and 5. When, the class dimensions in Q_c are the same of C_t (set of class to t_j in $\mathcal{T}$), the strategy *CCkNN* is the one that presents the best trade-off between the number of distance computations and memory space. However, if we observe the graphs regarding the estimated memory (Figures 5 (e) and (f)), the *CCkNN* requires more memory, e.g., in the experiments it required 5 times more than the *Multiple-InvIndex* regarding to the



(a) uRand500_16d_3ct_2c (b) uRand500_16d_3ct_3c (c) uRand500_32d_3ct_2c

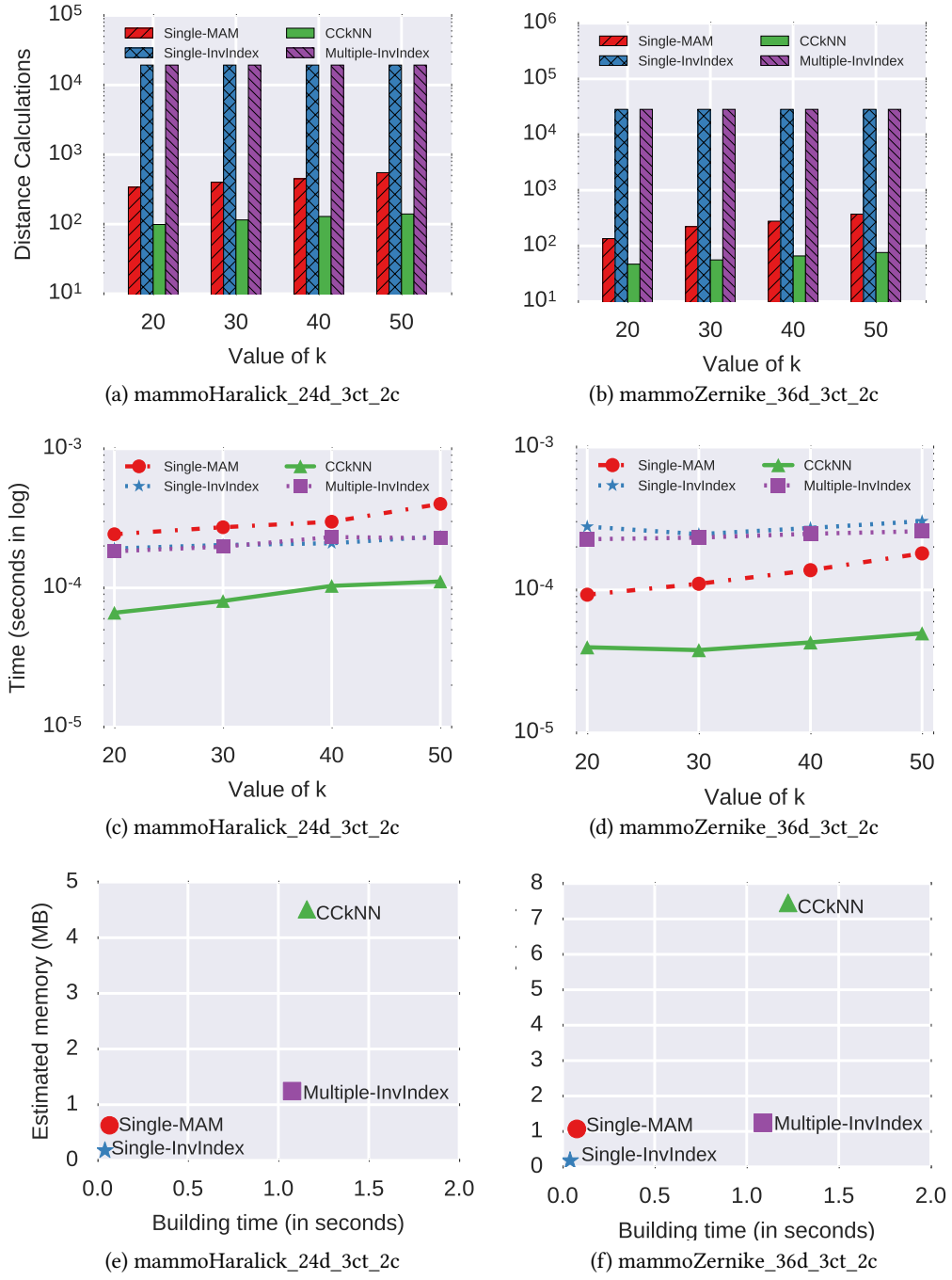
(d) gRand500_32d_3ct_3c (e) gRand500_32d_4ct_2c (f) gRand500_32d_4ct_3c

Figure 3: Number of distance calculations required by each method (number of distance calculations are in log scale). CC*k*NN achieved the best results for all datasets. For instance, with *gRand500_500_32d_4ct_3c* (f), *CCkNN* performed up to 2 orders of magnitude less distance calculations than Inverted-Index for all values of *k*.



(a) gRand500_32d_3ct_3c (b) gRand500_32d_4ct_2c (c) gRand500_32d_4ct_3c

Figure 4: Total time to execute the nearest neighbor queries, synthetic datasets with Gaussian distribution (a) in a 3-d unit hypercube, (b) and (c) in a 4-d unit hypercube.



(a) mammoHaralick_24d_3ct_2c (b) mammoZernike_36d_3ct_2c

(c) mammoHaralick_24d_3ct_2c (d) mammoZernike_36d_3ct_2c

(e) mammoHaralick_24d_3ct_2c (f) mammoZernike_36d_3ct_2c

Figure 5: Graphs of the analysis of efficiency for the real datasets. (a) and (b) Number of distance calculations performed by each method to execute the nearest neigbor queries. (c) and (d) Total time to execute similarity queries. (e) and (f) Estimated memory to keep each structure in-memory (x-axis) and building time for each one (y-axis).

mammoZernike_36d_3ct_2c (with 36 dimensions) dataset. Nevertheless, when compared to *Multiple-InvIndex*, it is possible to verify that *CCkNN* spends a shorter total time to execute the queries (see Figures 5 (c) and (d)).

When examining the results obtained with real and synthetic datasets (see Figures 3 and 5), it is possible to verify that the number of distance computations performed by the *CCkNN* remained up to two orders of magnitude smaller, when compared with the other

strategies. In general, *CCkNN* can employ strategies to reduce the memory space demand. For instance, it build the indexing trees with class combination smaller-fine-grained, i.e., keeping in-memory all class combinations where $|C_t| < |\mathfrak{C}|$. In fact, it is worth noting that when the number of class dimensions in $|Q_c| = C_t$, *CCkNN* is the best option to execute class-constraint k-nearest neighbor. However, if there are not labeled indexing trees regarding the class combination in $|Q_c|$, the best option will be to execute the queries on an indexing tree t_j having $|C_t| = |Q_c| - 1$ class dimensions. In this manner, *CCkNN* will execute queries on only one index, instead of on several ones. Notice that, besides being costlier to execute queries on indexing trees built over fine-grained class combinations when $|Q_c| < |C_t|$, due to the need of searching on all indexing trees that has the class dimensions in $|Q_c|$, it is also required to execute a merge of the results of each individual tree. Besides that, for this issue the literature presents studies about decreasing the number of metric indexes to reduce the total time to execute similarity queries [10]. In this context, the research presented in this paper is related to labeled datasets and the impact of applying constraints on k-nearest neighbor regarding the specific classes of the objects. In this study, we analyzed the impact of keep all the class combinations to execute the queries according to the Q_c.

7 CONCLUSIONS

When thinking of supporting efficient class-constraint k-nearest neighbor queries on large complex datasets, it is important to provide means not only to accelerate the building time of the structures, but also to employ strategies that do not degrade the effectiveness of similarity queries. The indexing structures analyzed herein, regarding the indexing of labeled datasets, met these requirements. In this study, we evaluated indexing structures based on complex data with multiple class dimensions. We applied concepts of combinatorial analysis [4] (such as combinations of class dimensions to create distinct subsets of classes) for building multiple indexes. Through our experimental analysis, we can state that the class-constraint k-nearest neighbor query *CCkNN*, has the following contributions:

- *Constrained similarity queries: CCkNN* restricts the queries to specific classes. In fact, this strategy was guided by the hypothesis that it is possible to apply class-constraints to a result of traditional k-nearest neighbor queries and still get a timely query processing. However, to be realistic our strategy requires more memory space, but it computes fewer distance computations. Thus, if your cost bottleneck is processing, go for *CCkNN*. In this manner, future works shall include the identification of the best class combinations to build the indexes. Therefore, the memory space demand will be reduced.
- *Speed-up the class-constraint querying process:* The strategy adopted by *CCkNN* queries takes advantage of employing multiples structures for speeding-up the query time, without adding extra weight on the processing.
- *Increase of query quality:* Our *CCkNN* aims at considering the query responses in a given context regarding the classes of the objects. This strategy restricts the results of the traditional k-nearest neighbor queries to the desired classes.

The strategy employs no classifier to help in the process. According to the experiments showed, *CCkNN* has the potential to speed up query processing by up to two orders of magnitude faster than the competitors and future work will investigate more intelligent ways to select the indexes that need to be built. Our proposal enables distributed query execution since the indexes are independent of each other. Hence, a future work is to evaluate this scenario.

ACKNOWLEDGMENTS

The author Jessica A. de Souza would like to thank Lucas de C. Scabora of GBDI (The Images and Data Bases Group at University of Sao Paulo) for his contribution on the paper and the Databases and Information Systems research group at TU Kaiserslautern which is headed by Prof. Sebastian Michel. This study has been supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and by RESCUER Project under grants EU 614154 and CNPq/MCTI 490084/2013-3.

REFERENCES

[1] T. Abeywickrama, A. M. Cheema, and D. Taniar. 2016. k-Nearest Neighbors on Road Networks: A Journey in Experimentation and In-Memory Implementation. In *Proceedings of the VLDB Endowment*, Vol. 9 (6). 492–503.
[2] P. Ciaccia, M. Patella, and P. Zezula. 1997. M-tree: an efficient access method for similarity search in metric spaces. In *Proceedings of the International conference on very large data bases (VLDB)* (15th ed.). 426–435.
[3] H. Ferhatosmanoglu, W. G. Aref, and M. H. Ali. 2001. Constrained Nearest Neighbor Queries. In *Proceedings of the International Symposium on Spatial and temporal Databases (SSTD)*. 257–276.
[4] S. Roberts Fred and Tesman Barry. 2009. *Applied Combinatorics.* Chapman & Hall/CRC 2nd ed.
[5] M. Heath, K. Bowyer, D. Kopans, R. Moore, and W. P. Kegelmeyer. 2001. The digital database for screening mammography. In *IWDM, Medical Physics*. 212–218.
[6] G. R. Hjaltason, D. S. Kaster, and H. Samet. 2000. *Incremental similarity search in multimedia databases.* Report CS-TR-4199. University of Maryland, College Park.
[7] M. Lee, D. Choi, S. Kim, H. Park, S. Choi, and C. Chung. 2016. The direction-constrained k nearest neighbor query - Dealing with spatio-directional objects. *Geoinformatica* (2016), 471–502.
[8] G. Navarro and N. Reyes. 2016. New dynamic metric indices for secondary memory. In *In Journal Information Systems*. 48–78.
[9] W. Ni, M. Gu, and X. Chen. 2016. Location privacy-preserving k nearest neighbor query under user's preference. *Knowledge-Based Systems* (2016), 19–27.
[10] P. H. Oliveira, L. Scabora, M. Cazzolato, W. D. Oliveira, A. J. M. Traina, and C. Traina-Jr. 2017. Efficiently Indexing Multiple Repositories of Medical Image Databases. In *the 30th IEEE International Symposium on Computer-Based Medical Systems*. IEEE.
[11] H. Samet. 2006. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmman.
[12] A. Thomasian. 2015. Singular Value Decomposition, Clustering, and Indexing for Similarity Search for Large Data Sets in Hight-Dimensional Spaces. *Big Data Algorithms, Analytics, and Applications* (2015), 40–68.
[13] Jr. C. Traina, A. J. M. Traina, C. Faloutsos, and B. Seeger. 2002. Fast indexing and visualization of metric datasets using Slim-trees. In *Proceedings of the IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 14 (2). 244–260.
[14] P. Zezula, G. Amato, V. Dohnal, and M. Batko. 2006. *Similarity Search: The Metric Space Approach.* Vol. 32. Springer.