ORTree: Tuning Diversified Similarity Queries by Means of Data Partitioning

João Victor de Oliveira Novaes[1](✉), Lúcio Fernandes Dutra Santos[2], Agma Juci Machado Traina[1], and Caetano Traina Jr.[1]

[1] Institute of Mathematics and Computer Sciences, University of São Paulo, São Carlos, SP 13566-590, Brazil
novaes.jvo@usp.br, {agma,caetano}@icmc.usp.br

[2] Federal Institute of Technology of North of Minas Gerais, Montes Claros, MG 39404-058, Brazil
lucio.santos@ifnmg.edu.br

Abstract. As modern applications gather more and more data, the data types also become more complex. Traditional retrieval operations based on identity and order comparisons are not suitable for those types. Instead, similarity operators are much more interesting for querying complex data and are gaining increasing attention. Similarity queries retrieve the elements most similar to a query center but, they tend to return elements that are very similar to others in the result set, reducing users' interest in the answer. To overcome this problem, researchers have considered incorporating a diversity degree in the similarity operators. Unfortunately, diversified similarity queries are computationally expensive, as they need to assess the relationship between each pair of elements in the result. Several works in the literature present techniques to speed up diversity in similarity queries, but they are either not scalable or only consider the diversity property. In this paper, we propose an index data structure, called the Omni-Range Tree (ORTree), that partitions the query space into a small subset of similar elements to a query element and prospect representative candidates aiming at dispatch diversified similarity queries. Our experimental evaluation shows that our index structure can reduce the query execution by time up to 95% without harming the quality of the results concerning other literature methods.

Keywords: Diversified similarity queries · Metric spaces indexing · Pivot-based space partitioning

1 Introduction

With the evolution of data acquisition and of the applications domains employing Database Management Systems (DBMS), it has become needed to store and retrieve more complex data, such as images, audio, videos, and long texts. Classic

FAPESP (grants No. 2016/17078-0, 2020/07200-9), CAPES (grant 001) and CNPq.

comparisons performed by current DBMSs are mainly based on identity and order relationships, which are not adequate for complex data. On the other hand, similarity queries are the information process that evaluates a element given by the user, called the query center (s_q), and retrieves of a set of elements that are alike, but not equal, to the reference element (s_q) [9,10].

The metric space model [8] is an efficient approach for handling similarity queries, in which data elements are mapped into a known domain where they are compared by a distance function (δ) to assess how similar are the elements, assuming that smaller distances correspond greater similarity among elements. The two most useful similarity retrieval operators are the similarity range (R_q) and the k-nearest neighbor (kNN_q) queries. A range query retrieves the elements from the dataset that are farther apart than a radius (r) from the query center s_q. A k-nearest neighbors query retrieves the k elements most similar to s_q. However, when similarity criteria are applied to large datasets with high cardinality and density, they tend to lose expressiveness and, consequently, quality. A major semantic-driven problem related to increasing data volume is that the similarity query operators are unable to filter the result set elements similar to each other [4]. The problem with too similar objects in result sets is that they can mislead users to believe that the database does not store the required information [4,9,11,14].

Several researchers have considered including diversity in query results, aiming at returning elements that are both similar to the query center and diverse from each other. Several diversification strategies can be found in the literature. They are classified as based on coverage or on novelty (also called as distance-similarity) [9]. The former return elements enforcing a dissimilarity threshold, returning only elements that respect a given distance between them. In this approach, the goal is to find elements that cover different information. The later search for elements that maximize a double criteria objective function, where similarity and diversity are balanced according to a user's defined preference parameter. Their goal is to find elements that are not redundant with the elements already found [4,14,16]. Depending on the data domain, one approach may be more suitable than the others although are important for result diversification. This work focuses on novelty-based diversification approaches.

Searches for diversified similarity are intrinsically costlier than searches seeking only similarity. This is due to two facts: more elements need to be loaded and compared, and each candidate needs to be compared not only to the query center but also to the other elements already in the answer. Novelty-based algorithms usually consider diversification as an NP-Hard optimization problem [4,14], but an exact solution is not usually obtained in a feasible time. An alternative to reduce computational costs relies either on heuristics or on metaheuristics [10]. However, both algorithms have scalability problems. The most common approach [11,14,15] is to select a subset of candidates to be processed by the diversification algorithms. However, not only the execution time but also the answer quality is directly impacted on how such selection is performed [11,15]. One of the most impacting problem of these approaches occurs when elements that do not maximize the objective function are selected. This often happens when applied approaches focus only on diversity [5,15].

In this work we present a strategy and corresponding algorithms to speed up similarity with diversity based on novelty. We seamlessly integrated an index structure with candidate selection methods to allow selecting elements considering both similarity and diversity. Our main contributions are: (***i***) An index structure that partitions the search space considering the distance among elements. (***ii***) Two algorithms to speed up similarity with diversity query algorithms to be employed in conjunction with our structure.

The remainder of this paper is organized as follows. Section 2 presents related works and basic concepts. Section 3 presents our proposal for space partitioning and selecting elements. Section 4 presents the evaluation environment and the results obtained. Finally, Sect. 5 presents our conclusions.

2 Background

Here we present related index structures and existing works, which allows partitioning elements of the dataset to perform similarity queries efficiently and foster obtaining diversified results.

2.1 Range-Tree

The Range-Tree (RT) [3,15] aims at quickly find the elements contained within the range of the query. It partitions the dataset considering the full range of values in every dimension. Figure 1a illustrates a RT storing a two-dimensional data, where the range of the first dimension spans from 0 to 2. In this structure, the root stores all elements within the range of the first dimension. The other non-leaf (intermediate) nodes store the partitions generated by the subranges of the root. To handle spaces with dimensionality greater than one, each node has a child pointing to a subrange tree that partitions the elements in the next dimension. During a search, whenever a node within the range of the current dimension is found, a search in the next dimension is performed, until there are no more dimensions to search. A d-dimensional range query Rd_r is expressed as a sub-range for each dimension of the dataset, such as $Rd_r = \{x_{init}, x_{finish}, y_{init}, y_{finish}\}$[1]. The time complexity to query a RT is $O(\log^d(n))$ and its space complexity is $O(n \log^d(n))$, where d is the dataset dimensionality and n is its cardinality.

2.2 MAM - Omni-Technique

A similarity query can be executed by a sequential scan, where every element is compared to the query center s_q. However, calculating every distance slow down the process, due to the high computational cost of similarity calculations. To

[1] Notice that a Rd_r correspond to elements within a sequence of values in each d-Dimension, thus it is distinct from a similarity range query R_q.

speed up the queries, many Metric Access Methods (MAM) were proposed to store and efficiently retrieve data based on the Metric Spaces properties, which allow reducing the number of similarity comparisons [8,13]. Among the several well-known structures we highlight those based on the Omni-Technique [13].

Given a dataset with a fractal dimension $\mathbb{D}$, the Omni-Technique is a pivot-based indexing approach that assumes that $\lceil \mathbb{D} \rceil$ is the ideal cardinality for a set of pivots $\mathcal{P}$ employed to accelerate query execution. The omni-technique aims at pre-computing the distances of every element to every pivot. The distances are called the omni-coordinates of each data element and they are employed to reduce the number of distance comparisons during a query execution. The process has two steps: filter and refine. During the filtering step, the omni-coordinates of the query center s_q is calculated and the triangular inequality property is used to find the regions that contain the query results, which can include false positives. The refinement step removes the false positives and generates the final answer.

The omni-coordinates generate a new search space, more compact than the original one. Thus, they can be used both as an indexing and a dimensionality reduction strategy. We take into account both benefits for developing our proposed method, as described in Sect. 3.

2.3 The Diversity Problem

A diversified similarity query can be defined as an optimization problem that looks for elements R that are both similar to the query center but also diverse from each other. This goal can be expressed as a double-criteria objective function that targets to maximize similarity and diversity, as follows. Given a dataset S, a query element s_q, an integer k, a function δ_{Sim} that measures how similar each element s_i is from s_q, and a function δ_{div} that measures how diverse two elements are, the diversification problem can be expressed as [9,11,14]:

$$R = argmax(\mathcal{F}(s_q, R)), \forall R \subseteq S : |R| = k, \tag{1}$$

$$\mathcal{F}(s_q, R) = (1-\lambda) \cdot \sum_{i=0}^{k} \delta_{sim}(s_q, r_i) + \frac{2\lambda}{(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \delta_{div}(r_i, r_j) : r_i, r_j \in R \tag{2}$$

Parameter $\lambda[0, 1]$ defines how much diversity the user expects. When $\lambda = 0$, the problem is reduced to a kNN_q. When $\lambda > 0$, the problem becomes NP-Hard with time complexity $O(n^k)$ (where $n = |S|$), as it must evaluate every subset $R(|R| = k)$ to find the one with the largest $\mathcal{F}$.

Several approximate algorithms have been proposed to generate a good query answer in a feasible time, some executing in $O(n^2)$ time. However, even considering this complexity reduction, they still can take a long time to get an answer. Thus, approximate algorithms typically have two phases: Candidate selection and Diversification. The candidate selection phase extracts a subset S' with cardinality $m = |S'| << |S|$. In this way, the search space is reduced to $m = |S'|$. In the diversification phase, a similarity with diversity algorithm is applied to the set of candidates S' [9,11,14].

2.4 Diversity Algorithms

One of the first and best-known diveristy algorithm in the literature is the *Maximal Marginal Relevance* (MMR) [2]. An element is marginally more important if it is as similar to the query element as it is diverse from the elements already inserted in the answer set. Thereafter, other elements that maximize the MMR function are incrementally inserted. The time complexity of MMR is $O(kn)$, however, other algorithms can generate better results.

The *Greedy Marginal Contribution* (GMC) [14] is an incremental algorithm that basically follows the same steps of MMR but uses another objective function, the *maximum marginal contribution* (MMC). The MMC function evaluates the contribution of the element $s_i \in S$ considering the similarity between s_i and s_q, the diversity between s_i and the elements already in R and the diversity between s_i and the elements of the candidate set S' that are not yet in R.

The *Greedy Randomized with Neighbor Expansion* (GNE) [14] is based on the GRASP meta-heuristic (*Greedy Randomized Adaptive Search Procedure*). The GNE can be divided in two phases: construction and local search. In the construction phase, the algorithm iteratively generates an initial solution to maximize MMC. In the local search phase, the algorithm improves the initial solution, looking for a higher quality solution in the neighborhood of the current solution. If no better solution is found, the current one is returned.

The *Max-Sum Dispersion* (MSD) [7] algorithm incrementally builds the answer R, selecting the pair of elements that maximizes the objective function. Basically, at each iteration it chooses two elements s_i and $s_j \in S$ that are both similar to s_q and different from each other. For cases where k is odd, MSD randomly chooses the last element to be inserted.

GMC, GNE, and MSD are capable of generating better results than MMR. However, their time complexity are $O(n^2)$, which makes the process of analyzing many elements even longer [11,14].

2.5 Candidate Selection

Several approaches to select/filter elements were developed to improve efficiency whereas also finding good answers. The most common use a similarity search (kNN_q or R_q) to return the subset S' with the m elements closest to s_q, but other candidate selection strategies have been considered too [5,11,14,15].

For example, in [11] were conducted an evaluation of distinct filter approaches combined to novelty algorithm, in which RDI standed out. RDI, returns m elements using the concept of Result Diversification based on Influence [12]. Although very fast, it is based on a method that does not guarantee that the selected elements actually maximize the objective function ($\mathcal{F}$). Another point is that RDI does not restrict the search space, so it is not uncommon that the entire dataset is analyzed, which can sometimes make the selection process slower than other approaches.

A modification of the algorithm for the Cover-Tree construction [1] was presented in [5], which here we call CT, aiming at efficiently find diversified sets

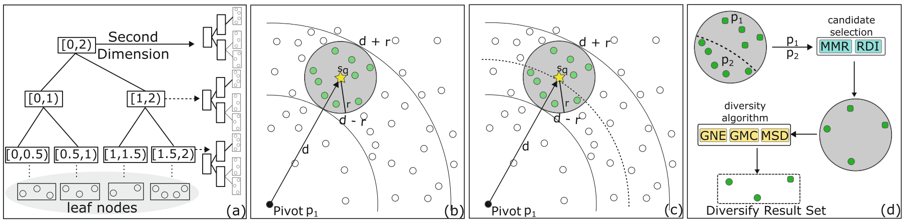

Fig. 1. Two-dimensional Range Tree and the process of candidate selection to partition the search space using one pivot. (a) Each node in the first dimension leads to another Range Tree in the second dimension. (b) Range query defined by $R_q(s_q, r)$. (c) The query range is partitioned (dotted line), generating two subranges. (d) Considering the generated partition, a candidate selection approach is applied on each partition. The elements selected from each partition are joined for the diversity algorithm.

in data streams. The proposed algorithm transforms each level of the Cover Tree into a possible solution for a diversification heuristic. Following up, several search algorithms were proposed, one of them returns the k elements contained in one of the upper nodes of the Cover Tree, which can be quickly obtained. However, the proposed algorithm builds the tree considering only the diversity between the elements. Furthermore, the construction algorithm has complexity $O(n^2)$, which makes the whole process as expensive as the algorithms previously presented.

The RC-Index [15] selects candidates using two data structures: a Range Tree and a Cover Tree. The Range Tree partitions the dataset and, for each partition, creates a corresponding cover tree. Thereafter, given a search range, the cover trees within the query range are used to extract a subset of candidates. The candidates are extracted in a way similar to the CT approach, the main difference being that, given a level (L_k) that contains k elements, the candidates at some lower level are returned: by default, three levels below L_k. However, this approach is ineffective for high-dimensional datasets, since its building complexity is $O(\gamma^6 log^{d+1}(n))$, in addition, its space complexity is the same as the Range Tree: $O(n \log^d(n))$.

The next section presents a novel structure which, as the RC-Index, partition the search space also using a Range Tree, but coupled to the Omni-technique to reduce the space complexity, which allows a much faster similarity query execution. Distinctly from RC-Index, we also present an algorithm to convert a similarity range query into a distance range query, which can be executed in an RT.

3 Methodology

Here we propose an efficient method to answer diversified similarity queries, based on two concepts: Spatial partition and Candidate selection. The first aims at partitioning the data into small subsets, so that the elements in each subset are similar to each other and the cost of selecting diversified candidate elements is as small as possible. The second concept aims at quickly selecting the elements from each subset that can maximize both similarity and diversity. To partition the dataset, our approach extends the Range Tree with the Omni techniques, creating the **O**mni-**R**ange **Tree** (the ORTree). It partitions the dataset using omni-coordinates, which improves the similarity range query (R_q) efficiency. Moreover, as the amount of evaluated omni-coordinates is defined by the fractal dimension ($\mathbb{D}$) of the dataset, it almost always reduces the data dimensionality too and, consequently, redux the structure time and space complexity. The Range Tree is employed due to its low query complexity of $O(\log^{\mathbb{D}}(n))$.

Building a ORTree is very similar to build a RT, but instead of the original attributes, the omni-coordinates are used. Algorithm 1 show the process for building a ORTree. After choosing the pivots, the omni-coordinates of every element are calculated (lines 1–8) in time complexity of $O(n)$, generating a $\mathbb{D}$-dimensional space. Thereafter, a RT is built as follows. The elements are sorted following the first dimension and inserted into the root node (lines 9–10). At this point, the elements are partitioned into two subsets considering the median of the first dimension of the omni-coordinates array. Next, the 'left' and 'right' children are built recursively for the current dimension (lines 16–17) and the 'next' child for the next dimension (lines 18–20), repeating recursively until there are no more dimensions.

Algorithm 1. Building the ORTree

Input: Set of pivot elements $\mathcal{P}$, δ a metric distance function and the dataset S.
Output: ORTree.

```
 1: $\mathcal{O}_S \leftarrow \emptyset$            ▷ set of omni-coordinates
 2: for $\forall\ s_i \in S$ do
 3:     $\mathcal{O}_{si} \leftarrow \emptyset$
 4:     for $\forall\ p \in \mathcal{P}$ do
 5:         $coord \leftarrow \delta(s_i, p)$        ▷ coordinate corresponding to p.
 6:         $\mathcal{O}_{si} \leftarrow \mathcal{O}_{si} \cup coord$
 7:     end for
 8:     $\mathcal{O}_S \leftarrow \mathcal{O}_S \cup \mathcal{O}_{si}$
 9: end for
10: Sort($\mathcal{O}_S$, 0)
11: $ORTree.root = Construct(\mathcal{O}_S, 0)$
12: return ORTree
13: function CONSTRUCT(set, dim)
14:     if set.size == 0 then
15:         return NULL
16:     end if
17:     $node.left = Construct(left, dim)$
18:     $node.right = Construct(right, dim)$
19:     $dim = dim + 1$
20:     Sort(set, dim)
21:     $node.next = Construct(set, dim)$
22:     return node
23: end function
```

Given a $R_q(s_q, r)$, the ORTree retrieves the nodes that store elements within the query range. To obtain the RD_r range, the following steps are executed. The omni-coordinates of s_q and then the ranges for each dimension are generated, defining the search range r as the omni-coordinate of s_q (its distance to each pivot): *range* : $\{O_{sq_i} - r, O_{sq_i} + r\}$. The same procedure is applied to all other dimensions. Given the ranges, the query finds and returns the nodes contained within the ranges. The distances from s_q to each element in nodes is calculated and only those within r are returned: $\delta(s_q, s_i) \leq r$. Our approach is able to support both the $k - NN_q$ and R_q similarity queries, but here we are going to focus on R_q.

Aiming at achieving a selection of better candidates than those provided by other methods in the literature, we took into account the data structure provided by ORTree to create a novel process that allows a more efficient selection of candidates, maintaining a quality equivalent to the other approaches. Our method uses the nodes retrieved by an ORTree to extract a set of m elements from each node within the range defined by $R_q(s_q, r)$. Thus, we use the partitions generated by a ORTree to quickly select the candidate elements.

Figure 1 illustrates our strategy. Initially, a range query is performed using the ORTree (Fig. 1b), then, considering that the returned elements will be in different nodes (partitions) (Fig. 1c), we apply, at each node, an algorithm to select m different elements. The selected elements are joined to form the final candidate set, which is passed to the diversifying algorithm (Fig. 1d). One of the main advantages of this approach is that the number of comparisons tends to be much smaller, speeding up the selection algorithm. Based on this principle, we developed two methods(RT_MMR and RT_RDI), based on different algorithms, which can select the candidates, that are both similar to s_q and diverse from others.

The Range Tree MMR (**RT_MMR**) aims at using MMR to select the candidates. It is faster than any of the GMC, GNE, and MSD algorithms, although, it follows the same diversification strategy, allowing for select elements considering the diversity preference (λ). When using MMR, it is expected that the elements selected from each node, be in smaller quantity than it would originally be, but with equivalent diversity.

The Range Tree RDI (**RT_RDI**) seeks the candidate considering the influence-based diversification approach. As it is coverage-based, this approach tends to be very faster than MMR. However, this approach tends to analyze more elements than the previous approaches and therefore may be slower. To get around this problem, we use this approach on each of the nodes returned by ORTree, allowing reducing the number of comparisons between the elements, and thus making the selection faster. Unlike the original approach that may analyze the full dataset, our approach reduces the search space to at most $R_q(s_q, r)$.

Every approach selects m elements from each node (or all elements when the node has less than m elements), so the number of elements returned from each node is expected to be much less than the original amount. Consequently, the number of elements selected by the approaches tends to be smaller than that defined by the $R_q(s_q, r)$ approach.

4 Experiments

We performed several experiments to validate our proposal and evaluate whether the execution of the diversity algorithms were indeed faster and whether the quality of the results remains equivalent when compared to the base algorithms from the literature. Three datasets were selected for this purpose: US_cities, NASA [6] and Corel[2]. The datasets were selected to evaluate the behavior of the proposed approaches, exploiting data with different cardinalities and dimensionalities, mainly in relation to the fractal dimension, which impacts the construction and querying times. From each dataset, 50 elements were randomly selected to be used as queries centers following a hold-out strategy. Table 1 summarizes the information about each dataset and the query parameters. We define the query range (r) so that the number of elements analyzed in each dataset is approximately the same.

Table 1. Datasets statistics.

Dataset	$\|S\|$	d	$\mathbb{D}$	Pivot number	$\delta_{sim}/\delta_{div}$	Range	Elements retrieved	Dataset description
US_Cities	25,374	2	1.62	2	L_2	{2.0, **4.0**}	{560, 1860}	Geographic coordinates of American cities
Nasa	40,150	20	2.63	3	L_2	{0.6, **0.7**}	{716, 1514}	Feature vectors generated from NASA images
Corel	20,000	9	4.8	5	L_2	{1.9, **2.3**}	{680, 1572}	Feature vectors generated from common images

We compared the results of ORTree with the following approaches from the literature: *Rq*, RC-Index and CT. *Rq* uses all elements returned by $R_q(s_q, r)$. To ensure that the elements returned are within the same range, the RC-Index was implemented using the ORTree instead of the original Range Tree. CT is built using the results of a $R_q(s_q, r)$ performed over the ORTree.

For each experiment, the following values were used as query parameters for the similarity with diversity queries (in bold are the default values): $\lambda = \{0.3, \mathbf{0.5}, 0.7\}$ and $k = \{10, \mathbf{20}, 30\}$. For the RT_MMR and RT_RDI approaches, we define by default that the number of elements to be selected from each ORTree node is the number of elements to be returned by the query, thus $m = k$.

4.1 Index Creation Time

Figure 2 shows the ORTree creation time compared to the RC-Index. Figure 2(a), shows the time for US_Cities and Nasa datasets. While Fig. 2(b) shows the creation time for the Corel dataset when we vary the number of pivots (1–5). Both figures show that the RC-Index has a much higher construction cost

[2] Sample extract from https://archive.ics.uci.edu/ml/datasets/corel+image+features, accessed at: 06/05/2022.

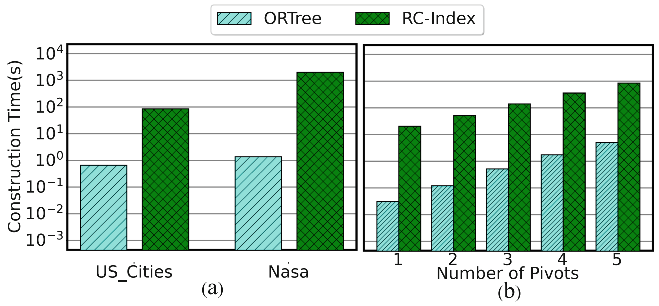

Fig. 2. ORTree and RC-Index build time in log scale for the US_Cities, Nasa and Corel dataset. (a) Time for US_Cities and Nasa. (b) Time for Corel dataset, with varying number of pivots.

than the ORTree. This is due to the difference in the complexity and the larger number of distance calculations performed by the RC-Index. It is also much slower than the ORTree, following the time complexity of $O(\gamma^6 nlog^{d+1}(n)) > O(nlog^d(n))$. Furthermore, the ORTree construction does not depend on distance calculations, whereas the RC-Index builds a cover tree, which requires several distance calculations, to build each node of the range tree, which greatly increases the execution time.

4.2 Quality Experiments

For each approach, the queries were performed using the GMC, GNE and MSD algorithms. Figure 3 shows the values returned by the objective function (Eq. 2) of each approach and algorithm. The Fig. 3 (a, b and c) show the search results for each of the algorithms using the US_Cities dataset for range = 4, Fig. 3 (d, e and f) show the results for Corel dataset with range = 2.4 and, Fig. 3 (g, h and i) show the results for Nasa dataset with range = 0.7.

To the US_Cities dataset, several approaches ties when $\lambda = 0.3$, but CT proved to be inferior to the other approaches. For $\lambda = 0.5$, all approaches are practically tied, except CT that achieves better results, for the GMC and GNE algorithms, including those generated by the traditional approach Rq. For $\lambda = 0.7$, we have a tie between the RT_MMR, Rq, and RC-Index approaches, the other approaches achieve lower results, with RT_RDI being better than CT. For the Corel dataset, again, many approaches tie, with CT achieving the worst results. For $\lambda = 0.5$, all approaches tie, and CT achieves better results for the GMC and GNE algorithms, but worse results for the MSD which is the best result in this case. At $\lambda = 0.7$ all approaches tie, except CT which achieves the worst results in both algorithms. In the Nasa dataset, for $\lambda = 0.3$, all approaches have very similar results, with CT showing lower results. For $\lambda = 0.5$, there is a tie between the approaches, making it difficult to point out an approach that is better in all cases. At $\lambda = 0.7$, the RT_MMR, Rq, and RC-Index approaches tie

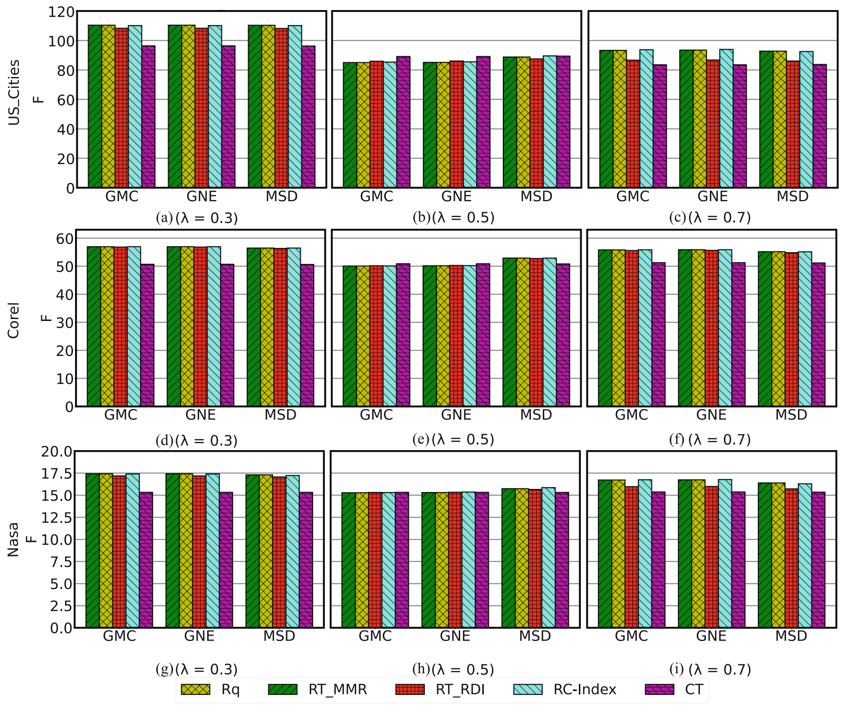

Fig. 3. Quality results according to the diversity objective function ($\mathcal{F}$), for dataset US_Cities (a, b and c) with $r = 4$, Corel (d, e and f) with $r = 2.3$ and Nasa (g, h and i) with $r = 0.7$

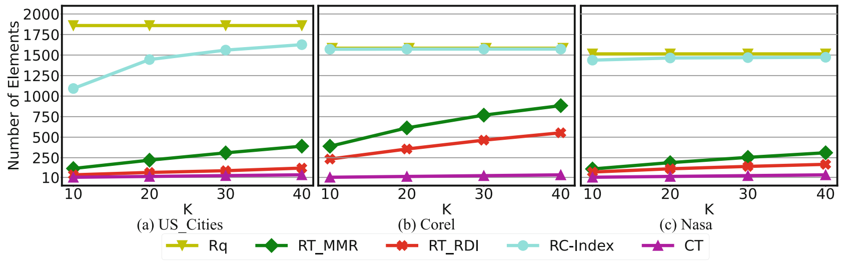

Fig. 4. Number of elements retrieved. (a) Number of elements for dataset US_Cities with $r = 4$. (b) Number of elements for dataset Corel with $r = 2.3$. (c) Number of elements for Nasa dataset with $r = 0.7$

in the results achieved, RT_RDI and CT achieve inferior results, with CT being the worst approach between the two.

In some datasets, the CT approach was able to generate better results than the traditional approach. In this case, the candidate selection process is likely to remove some solutions that are local optimal. Therefore, the candidate selection process can not only reduce the execution time of the algorithms but also provide higher quality. However, candidate selection can also remove optimal solutions. Therefore, the RT_RDI and CT approaches to achieve better results in some cases and worse in others. This situation did not happen with the RT_MMR and RC-Index approaches, they remain equivalent to Rq.

4.3 Number of Elements Retrieved

Figure 4 shows the average number of candidates selected by each approach. Figure 4a show the results for dataset US_Cities, Fig. 4b show the results for Corel dataset and Fig. 4c show the results for dataset Nasa. *Rq* always select the fixed number of elements, defined by a parameter. For all datasets, the number of candidates returned by our approaches (RT_MMR and RT_RDI), like the RC-Index, grows as k grows. However, in every case, our approaches retrieve fewer elements than Rq and RC-Index. For the US_Cities dataset (Fig. 4a), RT_MMR and RT_RDI always retrieve less than 30% of the elements from *Rq*. The RC-Index, on the other hand, retrieves around 50% fewer elements but as k grows, this value drops by approximately 80%. For the Corel dataset (Fig. 4b), the results are similar, with our approaches retrieving 57% fewer elements. However, RC-Index retrieves almost the same amount of elements as *Rq*, (discussed later). The Nasa dataset exhibits the same behavior of the other datasets, with RT_MMR and RT_RDI recovering less than 25% of the elements in relation to *Rq*, while RC-Index recovers closely the same amount. In all datasets, CT retrieves fewer elements because it always returns k elements.

Regarding the number of candidates returned by the RC-Index, because of the strategy of selecting the three-level elements below the cover trees (Sect. 2), number of elements is always greater than k. Also, depending on the distribution of the data, going down three levels may be enough to select all elements of the node. Another point that contributes to this situation is that the more pivots, the more partitions in the search space, which implies fewer elements per node in the ORTree (and RT). However, this situation does not happen with RT_MMR, and RT_RDI as they always return k or fewer elements from each node.

4.4 Query Time Evaluation

Figure 5 shows the execution time of the query algorithms using the proposed approaches. The figures show respectively the run-time for US_Cities (a and b), Corel (c and d) and NASA (e and f) datasets. The execution time using *Rq* and RC-Index tends to be longer than the execution time of every other approach, which is expected, due to the number of elements retrieved. In some cases, the RC-Index has a slightly higher cost than *Rq*, this happens because the RC-Index performs more operations and retrieves more or less the same number of elements as *Rq*. In all graphs, it is possible to see that RT_RDI is by far the fastest approach, followed by RT_MMR and CT approaches.

Figure 5 (a and b), shows that both RT_MMR and RT_RDI are faster than *Rq*, RC-Index, and CT. RC-Index is faster than *Rq* and CT faster than both. Figure 5

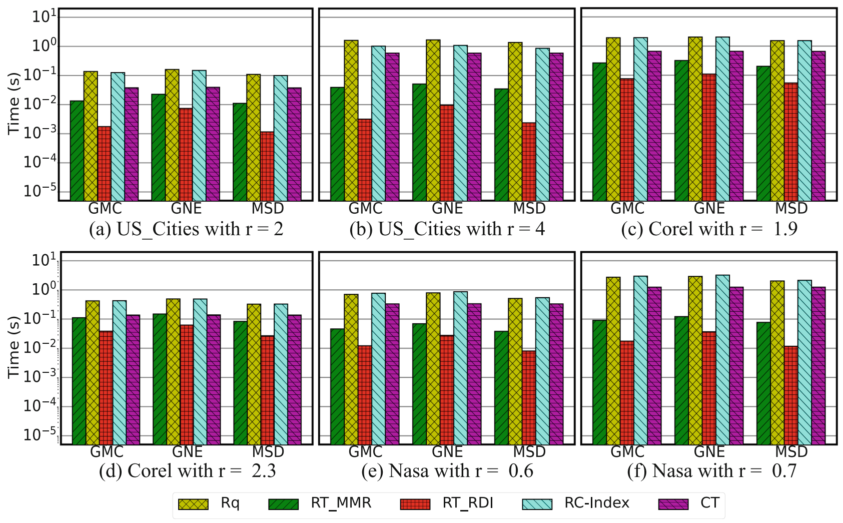

Fig. 5. Query time in log scale. (a - b) US_Cities, $r \in \{2, 4\}$. (c - d) Corel, $r \in \{1.9, 2.3\}$. (e - f) Nasa , $r \in \{0.6, 0.7\}$.

(c and d), shows that the results are similar to the previous figures, except that RC-Index has the same execution time as *Rq*, and in some cases (Fig. 5c), CT turns out to be faster than RT_MMR. In this case, the smallest amount of candidates compensate for the quadratic CT construction time. Finally, Fig. 5 (e and f), show that the execution time of RC-Index can be longer than that of *Rq*. For the other approaches, the results are as before, RT_RDI being the fastest, followed by RT_MMR and CT, with RT_MMR being many times faster.

The results show that RT_MMR and RT_RDI are by far the fastest approaches. regarding the Nasa dataset, the approaches are, respectively, 95% and 97% faster than the traditional approach (*Rq*).

5 Conclusions and Future Work

In this work, we present the ORTree, a new indexing structure based on the Range Tree and on the Omni-Technique, which allows performing diversified similarity queries much faster without reducing the quality of the answers. Along with this novel framework, we presented two approaches that use the ORTree to efficiently select candidates for the diversification process. Our experiments show that the proposed approaches can significantly reduce the number of elements that are analyzed in the process of diversification. Consequently, the query time was significantly reduced, in some cases being 95% faster. In addition, the quality results show that even reducing the number of candidate elements, the quality of the results remains equivalent to the traditional approach. Because of this, the experiments shows that ORTree is the best candidate selection approach for the analyzed aspects.

As a future work, we plan to develop new candidate selection approaches that use the ORTree and to develop alternative approaches for large search spaces. We also plan to extend the presented approaches, which are based on a similarity range query, to also handle the k-nearest neighbor queries.

References

1. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 97–104. ACM, New York, NY, USA (2006)
2. Carbonell, J., Goldstein, J.: The use of mmr, diversity-based reranking for reordering documents and producing summaries. In: Proceedings of the 21st SIGIR. pp. 335–336. ACM (1998)
3. De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry, pp. 1–17. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-540-77974-2
4. Drosou, M., Jagadish, H., Pitoura, E., Stoyanovich, J.: Diversity in big data: a review. Big Data **5**(2), 73–84 (2017)
5. Drosou, M., Pitoura, E.: Diverse set selection over dynamic data. IEEE Trans. Knowl. Data Eng. **26**(5), 1102–1116 (2014)
6. Figueroa, K., Navarro, G., Chávez, E.: Metric spaces library (2007). http://www.sisap.org/Metric_Space_Library.html
7. Gollapudi, S., Sharma, A.: An axiomatic approach for result diversification. In: WWW2009, pp. 381–390 (2009)
8. Hetland, M.L.: The basic principles of metric indexing. In: Coello, C.A.C., Dehuri, S., Ghosh, S. (eds.) Swarm Intelligence for Multi-objective Problems in Data Mining. Studies in Computational Intelligence, vol. 242, pp. 199–232. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03625-5_9
9. Novaes, J.V.O., et al.: J-EDA: a workbench for tuning similarity and diversity search parameters in content-based image retrieval. J. Inf. Data Manag. **12** (2021)
10. Lopes, C.R., Santos, L.F.D., Jasbick, D.L., de Oliveira, D., Bedo, M.: An empirical assessment of quality metrics for diversified similarity searching. J. Inf. Data Manag. **12**(3) (2021)
11. Santos, L.F.D., Oliveira, W.D., Carvalho, L.O., Ferreira, M.R.P., Traina, A.J.M., Traina, C.: Combine-and-conquer: improving the diversity in similarity search through influence sampling, Proceedings of the 30th SAC, pp. 994–999 (2015)
12. Santos, L.F.D., Oliveira, W.D., Ferreira, M.R.P., Traina, A.J.M., Traina, C.: Parameter-free and domain-independent similarity search with diversity. In: Proceedings of the 25th SSDBM. ACM, New York, NY, USA (2013)
13. Traina, C., Filho, R.F., Traina, A.J., Vieira, M.R., Faloutsos, C.: The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. VLDB J.l **16**(4), 483–505 (2007)
14. Vieira, M.R., et al.: On query result diversification. In: Proceedings of the 27th ICDE, 11–16 April 2011, Hannover, Germany, pp. 1163–1174. IEEE (2011)
15. Wang, Y., Meliou, A., Miklau, G.: RCIndex: diversifying answers to range queries. Proc. VLDB Endow. **11**(7), 773–786 (2018)
16. Zheng, K., Wang, H., Qi, Z., Li, J., Gao, H.: A survey of query result diversification. Knowl. Inf. Syst. **51**(1), 1–36 (2016). https://doi.org/10.1007/s10115-016-0990-4