Efficient Similarity Search in Metric Spaces with Cluster Reduction[⋆]

Luis G. Ares, Nieves R. Brisaboa, Alberto Ordoñez, Oscar Pedreira

Database Laboratory, Universidade da Coruña
Campus de Elviña s/n, 15071, A Coruña, Spain
{lgares,brisaboa,alberto.ordonez,opedreira}@udc.es

Abstract. Clustering-based methods for searching in metric spaces partition the space into a set of disjoint clusters. When solving a query, some clusters are discarded without comparing them with the query object, and clusters that can not be discarded are searched exhaustively. In this paper we propose a new strategy and algorithms for clustering-based methods that avoid the exhaustive search within clusters that can not be discarded, at the cost of some extra information in the index. This new strategy is based on progressively reducing the cluster until it can be discarded from the result. We refer to this approach as *cluster reduction*. We present the algorithms for range and kNN search. The results obtained in an experimental evaluation with synthetic and real collections show that the search cost can be reduced by a 13% - 25% approximately with respect to existing methods.

Keywords: similarity search, metric spaces, cluster reduction.

1 Introduction

Similarity search is a typical operation in many areas of computer science, such as pattern recognition, computational biology, multimedia information retrieval, or recommender systems, to name a few. Given a collection of objects and a function measuring the distance or dissimilarity between any two of them, similarity search finds the most similar objects to another one given as a query. The comparison of two objects is supposed to be computationally costly, so the goal of metric access methods (MAMs) is to solve the queries with the minimum number of distance evaluations.

Similarity search can be formalized through the mathematical concept of metric space. A metric space is composed by an universe of objects and a metric that determines the distance or dissimilarity between any two objects of that universe. Methods for searching in metric spaces preprocess the collection and build indexes that store precomputed information about the objects in collection.

[⋆] This work has been partially funded by "Ministerio de Ciencia y Innovación" (PGE and FEDER) refs. TIN2009-14560-C03-02 and TIN2010-21246-C02-01, and by "Xunta de Galicia" refs. 2010/17 (Fondos FEDER), and 10SIN028E.

This information is used during the search together with the properties of metric spaces to prune the search space and thus compare the query with a small portion of the objects in the collection [1–3].

The most studied types of query in metric spaces are range search and kNN search. *Range search*, $R(q, r)$, obtains all the objects up to a distance r from the query object q. *Near neighbor search*, $kNN(q)$, retrieves the k most similar objects to the query.

Methods for searching in metric spaces can be classified in pivot-based methods and clustering-based methods [1]. *Pivot-based methods* select a subset of objects from the collection to be used as pivots, and the index stores the distances from the pivots to the rest of objects. Given a query, the query object is compared with the pivots, and these distances are used to discard as many objects as possible without comparing them with the query.

Clustering-based methods partition the space into a set of disjoint clusters. Each cluster is represented by an object used as the cluster center. Each object in the collection belongs to the cluster corresponding to its closest center. The index maintains the information of each cluster, typically its center and its *covering radius*, that is, the distance from the cluster center to its furthest object in the cluster. Given a query, the query object is compared with the center of each cluster, and complete clusters are discarded if the information provided by the index determines that they can not contain objects in the result set. If a cluster can not be discarded, it is searched exhaustively, that is, the query object is compared with all the objects that belong to that cluster.

In this paper we present a new strategy and algorithms for precise metric-based search that avoid the exhaustive search within a cluster that can not be discarded by the index. Our proposal is based on the idea of defining regions within each cluster with respect to its center, in such a way that when we search within the cluster, it can be progressively reduced by discarding some of its regions, until the rest of the cluster is completely discarded. We refer to this strategy as *cluster reduction*, and it can be applied in any method using the covering-radius pruning criteria for discarding objects. Although we need to store more information in the index to maintain the regions within each cluster, we show that the space complexity of the index remains $O(n)$. We present the algorithms for range search and kNN search, and an experimental evaluation with real and synthetic collections of different nature that shows that this approach can improve the search performance by a 13% - 25% approximately with respect to existing methods. The results presented in the experimental evaluation also consider the effect of cluster reduction on the size of the index, and the trade-off between search cost improvement and space requirements.

The rest of the paper is structured as follows. Next Section briefly reviews related work on clustering-based methods for metric spaces. Section 3 presents cluster reduction and the algorithms for range and kNN search. In Section 4 we present the results we obtained from the experimental evaluation. Finally, Section 5 summarizes the conclusions of the paper and lines for future work.

2 Related Work

A *metric space* is a pair (X, d), where X is an *universe* of objects, and the function $d : X \times X \longrightarrow \mathbb{R}^+$ is a *metric* measuring the dissimilarity $d(x, y)$ between any two objects $x, y \in X$. The metric holds the properties of positiveness $(d(x, y) \geq 0)$, symmetry $(d(x, y) = d(y, x))$, and the triangle inequality $(d(x, y) \leq d(x, z) + d(z, y))$. The database or *collection* of objects is a finite subset $S \subseteq X$ of size $|S| = n$.

Clustering-based methods partition the space into a set of disjoint clusters, and the index stores the information about this partition. The information for cluster $\mathcal{C}_i$ includes at least the object used as the center of the cluster, c_i, the set of objects that belong to that cluster, and the covering radius, $cr_i = max\{d(c_i, x)/x \in \mathcal{C}_i\}$, that is, the distance from the center to its furthest object in the cluster. For each cluster $\mathcal{C}_i$, its center and its covering radius define a *ball* in the space, (c_i, cr_i), containing all the objects that belong to the cluster.

Given a range query $R(q, r)$, the query object is compared with each cluster center c_i, obtaining the distances $d(q, c_i)$. A range query defines a ball in the space, (q, r). For each cluster $\mathcal{C}_i$, the cluster can be discarded without comparing the query with its objects if $d(q, c_i) - cr_i > r$, that is, if the ball (c_i, cr_i) does not intersect the ball (q, r).

In the case of kNN queries, the distance $d(q, c_i) - cr_i$ gives us a lower bound on the distance from q to any of the objects that belong to the cluster $\mathcal{C}_i$, that is, $\forall x \in \mathcal{C}_i,\ d(q, x) \geq d(q, c_i) - cr_i$ (if the query object q falls within the ball defined by the cluster, (c_i, cr_i), the lower bound is negative, so it does not give us useful information). This lower bound gives us a hint of which clusters we should visit first and when to stop the search, that is, when none of the remaining clusters contains objects closer to q than its current k^{th} nearest neighbor.

The search complexity is given by the sum of the internal and external complexities. The *internal complexity* is the number of distance evaluations needed to compare the query object with the cluster centers. The *external complexity* is the number of distance evaluations needed to compare the query object with the objects in the clusters that could not be discarded.

Clustering-based methods build smaller indexes and behave better in high-dimensional spaces with respect to pivot-based methods. Existing clustering-based methods differ in how they partition the space, how that partitioning is reflected in the index structure, and on the criteria used for pruning the search space. Most methods partition the space in a recursive way and create a tree index that reflects that partition.

BST [4] recursively partitions each cluster into two clusters, and creates a tree index that maintains the information of the partition. In a first level, two objects are selected as cluster centers. The root of the index stores the center and covering radius of each cluster, and each cluster is then recursively partitioned following the same schema. GHT [5] partitions the space following the same schema, but changes the criteria to prune the tree during the search. In this case, the left subtree at each node is searched if $d(q, c_l) - r \leq d(q, c_r) + r$, and the right subtree is searched if $d(q, c_r) - r \leq d(q, c_l) + r$. GNAT [6] is a generalization of GHT in which more than two clusters are created at each node. In addition, each node stores the distances between the centers of the clusters, so some of them can be discarded without comparing them with the query object. VT [7] improves BST by using two or three centers in each node and storing in each new node the closest object from the parent node. SAT [8] creates a tree structure that approximates the *Delaunay graph* of the partition, and the search traverses the tree discarding complete clusters when possible.

The M-Tree [9] recursively partitions the space trying to obtain clusters as compact as possible to improve the search cost. This method established an important landmark since it supports the dynamics of a real database system. Its structure is suitable for efficient secondary memory storage, and it supports dynamic insertions and deletions of objects in the database without degrading the index performance, by rearranging the index on such operations, and adapting it to the new content of the database. The Slim-Tree [10] is a well-known modification of the M-Tree that reduces the overlap between the clusters.

All the methods described above create a recursive partition that generates a tree-like index. List of Clusters [11] follows a different approach and organizes the index as a list instead of as a tree. A first object is used as a cluster center for the first cluster. Once this cluster is full, the rest of the collection is processed in the same way. The size of the cluster is determined either by a fixed covering radius or by a fixed number of objects. The search traverses the list discarding the clusters when possible, and exhaustively searching non-discarded clusters.

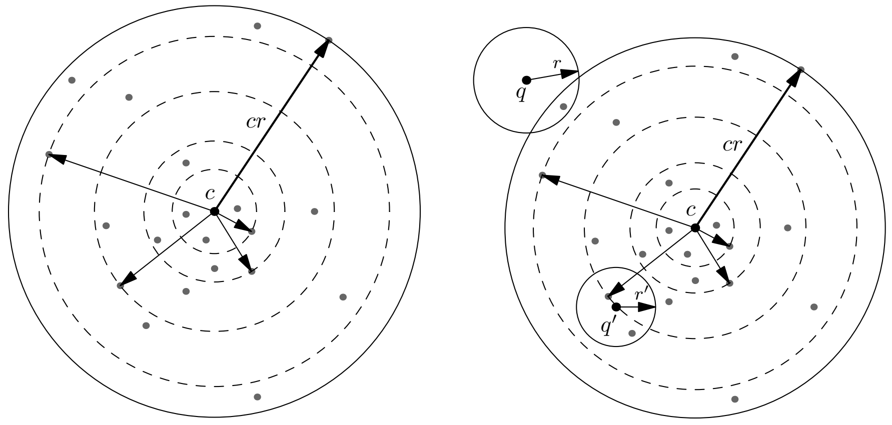
3 Similarity Search with Cluster Reduction

In this Section we present the idea of cluster reduction, and the corresponding algorithms for range search and kNN search. The goal of cluster reduction is to decrease the external complexity of the search by avoiding the exhaustive search within clusters that could not be discarded with the information of the index.

We present the algorithms for the particular case of List of Clusters, although this approach could be extended to other clustering-based methods using the covering radius pruning criteria.

3.1 Defining Intermediate Regions

Let $\mathcal{C}_i$ be a cluster with center c_i and covering radius cr_i, and let $\{x_{i1}, \ldots, x_{im}\}$ be the set of objects that belong to $\mathcal{C}_i$ (where m is the size of the cluster). These objects define a set of distances with respect to the center of the cluster, $\{d(c_i, x_{i1}), \ldots, d(c_i, x_{im})\}$. We select some of these distances to define intermediate regions within the cluster $\mathcal{C}_i$, in such a way that each selected distance acts like an additional internal covering radius. If we want to define β regions within each cluster, we have to select $\beta - 1$ distances. We select the distances in such a way that each region contains the same number of objects. If the number of regions, β, does not divide the number of objects in the cluster, one of the regions will have fewer objects than the others.



(a) Cluster divided in regions (b) Example of range search

Fig. 1. Range search with cluster reduction.

Figure 1(a) shows an example in which a cluster is divided into five internal regions. Each internal region contains four objects. The reason for selecting regions that contain the same number of objects is that in this way they adapt better to the distribution of the objects within the cluster. As previously stated in [1, 11], the distribution of the objects with respect to the center of the cluster is not uniform. Choosing regions with the same "width" could led to a region that contains almost all the objects in the cluster, which would be useless for the algorithms we present in this section.

The smallest number of regions we can define within a cluster is two, and, at most, we could define as many regions as objects in the cluster. In the first case, we have the chance of discarding half of the cluster without comparing it with the query. In the second case, each comparison of q with an object of the cluster gives us a chance of discarding the rest of the cluster without comparing it with the query. Therefore, the more the regions we define within clusters, the more chances of reducing the search cost.

However, the number of regions also affects the space requirements of the index. If we define as many regions as objects in the cluster, the space requirements would be almost the double, although the space complexity would still be $O(n)$. Therefore, there is a trade-off between the search cost and space requirements in terms of the number of regions. As we will see in the experimental evaluation, a small number of regions leads to a search cost that is very close to that obtained when the number of regions equals the number of objects.

3.2 Additional Considerations

Cluster reduction can be viewed as a generalization of the range-pivot distance constraint policy for discarding objects [2] if we consider the limit of each internal region as an additional covering radius. Similarly, cluster reduction can be viewed as a generalization of the object-pivot distance constraint for discarding objects [2], since the center of the cluster assumes a role similar to that of a pivot for the objects of that cluster. By knowing to which region each object in the cluster belongs, and the limits of that region, we have a range bounding the distance from the cluster center to each object, which is a case of range coarsening [1].

The idea of defining regions within clusters has already appeared in previous work. All methods that recursively partition the space divide each cluster into smaller clusters, although the leaf nodes have to be searched exhaustively. VPT [5] and MVPT [12] recursively partition each cluster into regions of the same size, which are further partitioned using the same schema. M-Index [13] assigns to each object a *key* that combines the identifier of the cluster to which it belongs, and its distance to the center of the cluster. This can be seen as a particular case of cluster reduction in which the distances from all the objects to the cluster center are stored. The PM-Tree [14] is an extension of the M-Tree in which each cluster is divided into *hyper-rings* with respect to a set of pivots.

The main differences of our proposal with previous approaches are: (*i*) we divide the cluster into several concentric regions with respect to the center (that is, without using other cluster centers or pivots) in an *onion-like* style, that is, each region is contained by another region; (*ii*) as we will see in the description of the algorithms for range search and kNN search, this allows us to process each cluster one region at a time until one of them allows us to discard the rest of the cluster; (*iii*) the results of our experimental evaluation show that using a small number of regions within each cluster leads to a search cost very close to that obtained when using as many regions as objects in the cluster.

Applying the cluster reduction strategy to an existing method modifies the structure of the index, since we need to store, for each cluster, which objects belong to each region, and also the distances from the center to the objects that define the limit of each region. These features of the index allow us to implement it using different data structures. Dynamic capabilities are also supported, allowing deletions (in general, logical deletions), and insertion of new objects. If the number of modifications in a cluster is high, restructuring the regions of the cluster in the index is necessary to maintain its search performance.

3.3 Range Search

The algorithm for range search proceeds initially in the same way as the original algorithm, only changing how clusters that can not be discarded are searched. Given a range query $R(q, r)$, the query object is compared with all the cluster centers c_i, obtaining the distances $d(q, c_i)$. For each cluster, we have three possibilities:

(a) $d(q, c_i) - cr_i > r$: in this case, the ball (c_i, cr_i) defined by the cluster $\mathcal{C}_i$ does not intersect the ball (q, r) defined by the query, so the whole cluster $\mathcal{C}_i$ is discarded from the result without comparing it with the query object.

(b) $d(q, c_i) - cr_i \leq r$, and not $d(q, c_i) + r \leq cr_i$: the ball (c_i, cr_i) defined by $\mathcal{C}_i$ intersects the ball (q, r) defined by the query. In this case, we can not discard the cluster. Instead of comparing the query object with all the objects in the cluster, we start the comparison at the outermost region of the cluster. In each region, we compare the query object with all the objects in that region. As we process each region, the cluster is being progressively reduced. The search within the cluster stops when the limit of the next region does not intersect the query ball. That is, we use the limit of each region as the covering radius of the cluster as we reduce it.

(c) $d(q, c_i) - cr_i \leq r$, and $d(q, c_i) + r \leq cr_i$: the ball defined by the cluster not only intersects the query ball, but it contains it. In this case, the rest of the clusters do not have to be explored. The search within the cluster starts at the outermost region that intersects the query ball, and continues until we reach a region that does not intersect it.

Figure 1 shows an example of range search with cluster reduction. The left part of the figure shows a cluster containing 20 objects, that has been divided into 5 regions, each of them containing 4 objects. The arrows that start at the center of the cluster point to the objects that define the limit of each region. The right part of the figure shows two range queries.

In the first case, $R(q, r)$, the query ball intersects the cluster but is not contained within it. Therefore, the search compares the query with the objects in the outermost region of the cluster. Once we have processed this region and therefore reduced the cluster, the next region does not intersect the query ball, so the rest of the cluster can be discarded. In this example, the cost of searching within the cluster is reduced to a 20% of its objects.

In the second case, $R(q', r')$, the ball defined by the cluster contains the query ball. Since the query ball only intersects the second and third regions of the cluster (counting from the outermost one), the search can be reduced to those regions. The search within the cluster is solved comparing only the query object with two of the five regions. In this case, the search stops because we do not need to explore any other cluster.

3.4 *k*NN Search

In the case of kNN search, the algorithm proceeds by visiting the most promising clusters first, as in the original algorithm. The difference is that in each step of the search we do not process the whole cluster, but only its closest region to q that has not still been processed. After processing that region, this cluster is reduced and the search continues with the next most promising cluster, including the cluster we have just processed and modified.

```
kNNSearch($q$)
foreach cluster $\mathcal{C}_i$ do
    $d_i \leftarrow d(q, c_i)$
    Insert($clustersQueue$, $\mathcal{C}_i$, $d_i - cr_i$)
end
$\mathcal{C}_i \leftarrow pull(clustersQueue)$
while $d(q, c_i) - cr_i < radius(neighborsQueue)$ do
    $Reduce(q, \mathcal{C}_i, neighborsQueue)$
    $Insert(clustersQueue, \mathcal{C}_i, d_i - cr_i)$
    $\mathcal{C}_i \leftarrow pull(clustersQueue)$
end
```

Algorithm 1: Pseudocode for kNN search with cluster reduction.

For each cluster $\mathcal{C}_i$, the distance $d(q, c_i) - cr_i$ gives us a lower bound on the distance from q to any object in $\mathcal{C}_i$, that is:

$$\forall x \in \mathcal{C}_i,\ d(q, x) \geq d(q, c_i) - cr_i \tag{1}$$

Given two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, the cluster $\mathcal{C}_i$ is more promising than $\mathcal{C}_j$ if:

$$d(q, c_i) - cr_i < d(q, c_j) - cr_j \tag{2}$$

Given a kNN query, the query object is compared with all the cluster centers, and the clusters are arranged into a priority queue, in such a way that the most promising clusters are processed first. In each step of the search, the most promising cluster is pulled from the queue, and the query is compared only with the objects in its closest region still not processed, updating the list of k candidate nearest neighbors as necessary. The cluster is reduced since we have processed one of its regions, and it is reinserted again in the queue with a new priority, resulting from the reduction of the cluster. This procedure is repeated until the lower bound of the distance from q to the next cluster to be processed is greater than the distance from q to its current k^{th} candidate neighbor.

Pseudocode 1 summarizes the algorithm for near neighbor search with cluster reduction. The algorithm uses two priority queues: *clustersQueue* stores the clusters according to how promising they are for the search, and *neighborsQueue* maintains in each step of the algorithm the k nearest neighbors of q among the objects that have already been processed (*neighborsQueue* has a limited capacity of k objects). The function *Reduce* (line 8) compares q with the objects in its closest region still not processed of $\mathcal{C}_i$, and sets the covering radius of $\mathcal{C}_i$ to the limit of its next region.

Figure 2 shows an example of 2NN search with two clusters. As we can see in the figure, the distance from q to $\mathcal{C}_1$, d_1, is smaller than the distance from q to $\mathcal{C}_2$, d_2, so $\mathcal{C}_1$ is more promising and it is processed first. The query is compared with the objects in the first region of $\mathcal{C}_1$, and the cluster is reduced and inserted again into the priority queue. Now, the distance from q to $\mathcal{C}_1$, d_3, is larger than

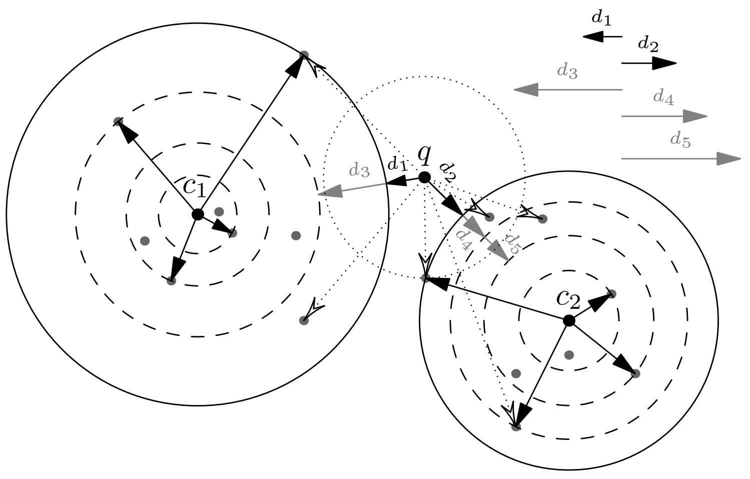


Fig. 2. kNN search with cluster reduction.

the distance from q to $\mathcal{C}_2$, d_2. In this point, the search stops processing $\mathcal{C}_1$ and jumps to $\mathcal{C}_2$, since it is more promising at this moment. After processing the first region of $\mathcal{C}_2$, the distance from q to $\mathcal{C}_2$, now d_4, is smaller than the distance to $\mathcal{C}_1$, d_3, so we process the next region of $\mathcal{C}_2$. After processing the second region of $\mathcal{C}_2$, the search stops, since d_3 and d_5 are greater than the distance from q to its second nearest neighbor (distance represented by the ball in dotted line). Note that the search has finished without having searched exhaustively within either $\mathcal{C}_1$ or $\mathcal{C}_2$, but only within one region of $\mathcal{C}_1$, and two regions of $\mathcal{C}_2$.

4 Experimental Evaluation

In this Section we present the results we obtained in the experimental evaluation of the new algorithms for range search and kNN search, considering both the effect of cluster reduction on the search cost and on the space requirements of the index.

We have implemented the new algorithms on a List of Clusters, under the framework provided by the SISAP Library[1]. In our experiments we used six datasets from the library:

- English: a collection of 69,069 words from the English dictionary.
- German: a collection of 75,086 words from the German dictionary.
- Nasa: contains 40,150 images from NASA archives represented by feature vectors of dimension 20.

[1] http://sisap.org/Metric_Space_Library.html

– Colors: contains 112,544 color histograms, represented by feature vectors of dimension 112.
– Uniform-10, Uniform-12: collections of 100,000 vectors of dimensions 10 and 12 respectively, with uniform distribution in the unitary cube.

We have chosen two collections of each type (words, images, and uniformly distributed vectors) since they contain objects of the same nature but present different complexities for the search. For each dataset, 90% of the objects were used as the collection to be indexed, and the remaining 10% were used as query objects. In the case of word datasets, objects were compared using the edit distance. In the case of vector datasets, we used the Euclidean distance.

4.1 Range Search

In order to evaluate the search cost obtained with cluster reduction for range search, we used cluster sizes $\{10, 20, 40, 60, 80, 100\}$, and values of β (number of regions) of 2, 5, 10, and 20 regions within each cluster, as well as the case in which we used all possible regions within each cluster. The search radius was adjusted to retrieve an average of 0.01% of the database for each query. Figure 3 shows the mean number of distance computations for processing all the queries (10% of the dataset) for each collection and method. In the figure, "`LC`" stands for List of Clusters without applying cluster reduction, "`LC CR-i`" stands for list of clusters applying cluster reduction with $\beta = i$ regions within each cluster. "`LC CR-all`" stands for List of Clusters using as many internal regions as objects in each cluster.

As we can see in the results, the application of cluster reduction produces a significant improvement on the search cost. The higher cost improvement is obtained when we use all the objects in the cluster to define a region. However, the results obtained when using a smaller number of regions are very close to the best result. An important result is that a significant part of the search cost improvement is obtained when using just 5 regions within each cluster. Adding more regions within each cluster reduces the search cost even more, but from the results we can see that when using 10 or 20 regions within each cluster, adding more regions does not improve significantly. Note also that this behavior holds for all the cluster sizes we have considered and for all collections, no matter their size or complexity.

4.2 *k*NN Search

In order to evaluate the performance obtained with cluster reduction for kNN search, we used values of k ranging from 1 to 10, and values of β of 2, 5, 10 and 20 regions. We also considered the case in which each object in the cluster defines an internal region. In these experiments, the cluster size was fixed to 40, which produced good results in the previous set of experiments for all collections. Figure 4 shows the results we obtained. The legend of the figures follows the nomenclature we used in the experiments for range search.

As we can see in the results, the behavior of the search cost for different values of β is similar to that obtained in the case of range search. The most significant improvement in the search cost is obtained when using just 5 regions within each cluster. When using 10 regions within each cluster, the search cost is very close to that obtained when maintaining the distances from the cluster center to all the objects in the cluster. The results are homogeneous for all values of k in all collections.

4.3 Search Cost and Space Requirements

As we explained in Section 3, the number of regions defined within each cluster produces a trade-off between the improvement in search cost and the increment in the space requirements of the index. We obtain the best result in terms of search cost when all the objects in the cluster are used to define a region within the cluster, but this almost doubles the space requirements of the index. However, as we have already seen, when using a smaller number of regions within each cluster, the search cost is very close to the best result, with a very reduced increment in the space requirements.

Table 1 shows for each collection the size of the index obtained when using 2, 5, 10, 20, and all possible regions within each cluster (in Kbytes). The column "Relative" shows the relative value of the size of the index with respect to the list of clusters without cluster reduction (shown in first row, LC). The increments in the size of the index range from a 2% to a 20% when the number of regions is between 2 and 10. When each object of the cluster is used to define a region in the cluster, the space overhead reaches a 89%. Although in the case of using all possible regions the space of the index is almost doubled, cluster reduction preserves the $O(n)$ space complexity of the index, an important property of clustering-based methods when compared with pivot-based methods.

Figure 5 shows results on the trade-off between the search cost improvement and the increment of the index size for range search queries. The improvement of search cost and the increment of the size of the index are expressed as relative values (in %) with respect to the search cost and index size of the original list of clusters (as in Table 1). The figure considers all collections and values of β in $\{2, 5, 10, 20, all\}$. When we define 2 regions within each cluster, the search cost is reduced a 10% approximately, with a space overhead of only a 2%. Using 5 regions produces a space overhead of 9%, while the search cost improvement ranges between 13% - 24% approximately with respect to list of clusters without cluster reduction, depending on the collection. When the number of regions is 10, the search cost improves slightly. From this point, using more regions within each cluster increments the size of the index but does not affect significantly to the search cost. This result is important since it shows that we only need to define a small number of regions within each cluster, independently of the size of the cluster.

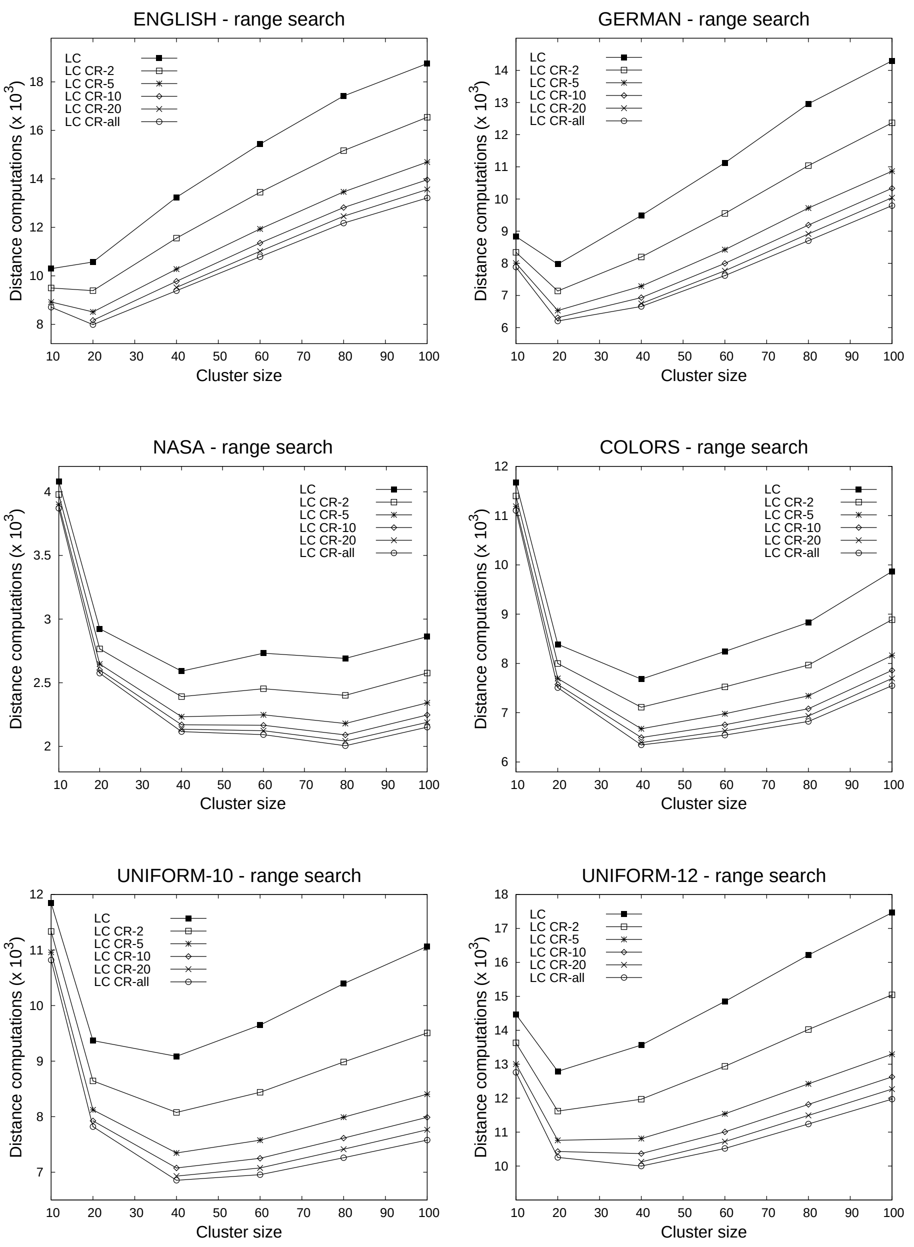


Fig. 3. Search performance in range search. Each figure shows the mean number of distance computations (in thousands of distances) in terms of the size of the clusters.

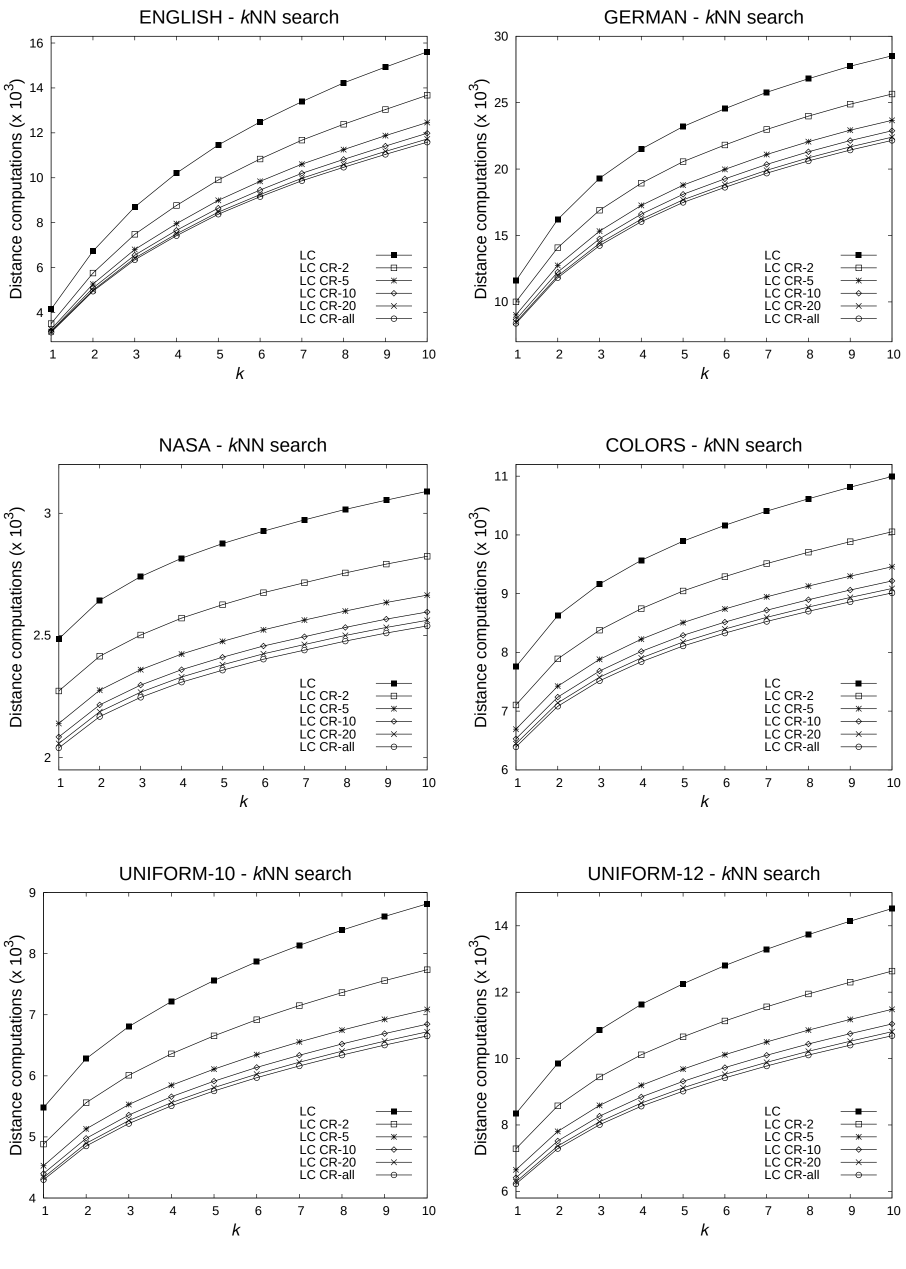


Fig. 4. Search performance in kNN search. Each figure shows the mean number of distance computations (in thousands of distances) in terms of k, the number of neighbors searched for each query object.

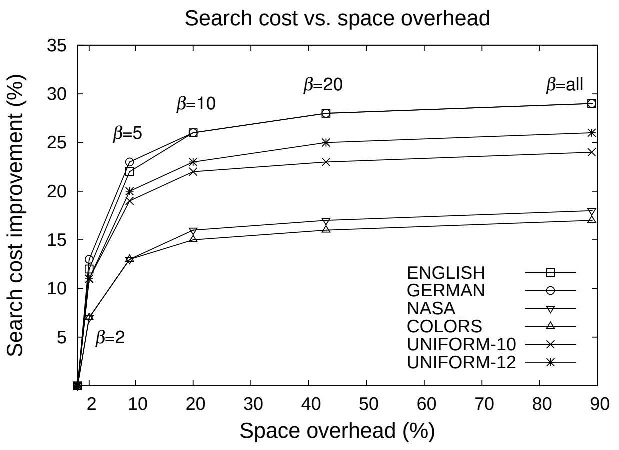


Fig. 5. Search cost vs. space trade-off in range search with cluster size 40.

Table 1. Effect on space requirements (Kbytes).

β	English	German	Nasa	Colors	Relative
LC	260.65	283.34	151.53	424.66	1.00
2	266.57	289.79	154.97	434.31	1.02
5	284.34	309.10	165.29	463.26	1.09
10	313.95	341.29	182.50	511.50	1.20
20	373.17	405.66	216.92	607.99	1.43
all	491.60	534.41	285.74	800.96	1.89

5 Conclusions

In this paper we have presented a new strategy, namely cluster reduction, and new algorithms that avoid the exhaustive comparison of the query with the objects in clusters that can not be discarded with the information in the index, thus reducing the external complexity. We have presented the algorithms for the particular case of List of Clusters, but this approach could be extended to other clustering-based methods using the covering radius pruning criteria.

We define internal regions within each cluster, in such a way that all regions contain approximately the same number of objects. When searching within a non-discarded cluster, the cluster is processed one region at a time, in order of proximity to the query. This allows us to progressively reduce the cluster until we can discard the rest of its regions. The definition of a reduced number of regions within each cluster reduces significantly the search cost, with a small increment in the size of the index, and maintaining the space complexity as $O(n)$.

We have presented an experimental evaluation with both real and synthetic collections from the SISAP Metric Spaces Library. Our results show that the improvement in search cost ranges between a 13% - 25% approximately depending on the collection and on the number of regions defined within the clusters. The results also show that the search cost when using a small number of regions is very close to that obtained when using all the possible regions.

Some aspects of this work remain as lines for future work. The most immediate, to test the strategy of cluster reduction with other methods using the covering radius pruning criteria, to compare it with other traditional MAMs, and to evaluate its scalability with massive data sets. We are also exploring how to extend this strategy to other similarity queries, such as the similarity join.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys **33** (2001) 273–321
2. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity search. The metric space approach. Volume 32 of Advances in Database Systems. Springer (2006)
3. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Transactions on Database Systems **28**(4) (2006) 517–580
4. Kalantari, I., McDonald, G.: A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering **9** (1983) 631–634 IEEE Press.
5. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters **40** (1991) 175–179
6. Brin, S.: Near neighbor search in large metric spaces. In: Procs. of Conf. on Very Large Databases (VLDB'95), Morgan Kaufmann Publishers (1995) 574 – 584
7. Dehne, F., Noltemeier, H.: Voronoi trees and clustering problems. Information Systems **12**(2) (1987) 171–175
8. Navarro, G.: Searching in metric spaces by spatial approximation. In: Procs. of String Processing and Information Retrieval (SPIRE'99), IEEE CS Press (1999) 141–148
9. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Procs. of Conf. on Very Large Databases (VLDB'97), ACM Press (1997) 426–435
10. Jr., C.T., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-trees: High performance metric trees minimizing overlap between nodes. In: Procs. of Extending Database Technology (EDBT'00). LNCS(1777), Springer (2000) 51–65
11. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters **26**(9) (2005) 1363–1376
12. Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. In: Proc. of the ACM Conf. on Management of Data (SIGMOD'97), ACM Press (1997) 357–368
13. Novak, D., Batko, M., Zezula, P.: Metric index: An efficient and scalable solution for precise and approximate similarity search. Information Systems **36**(4) (2009) 721–733
14. Skopal, T., Pokorný, J., Snásel, V.: Pm-tree: Pivoting metric tree for similarity search in multimedia databases. In: Procs. of Advances in Database Systems (ADBIS'04), Local Procs. (2004) 803–815