preprint of http://dx.doi.org/10.1016/j.ipm.2007.03.002

Nearest Neighbor Search in Metric Spaces through Content-Addressable Networks ⋆

Fabrizio Falchi [a], Claudio Gennaro [a,*], Pavel Zezula [b]

[a] *ISTI-CNR, Pisa, Italy*

[b] *Faculty of Informatics, Masaryk University, Brno, Czech Republic*

Abstract

Most of the Peer–to–Peer search techniques proposed in the recent years have focused on the single-key retrieval. However, similarity search in metric spaces represents an important paradigm for content-based retrieval in many applications. In this paper we introduce an extension of the well–known Content–Addressable Network paradigm to support storage and retrieval of more generic metric space objects. In particular we address the problem of executing the nearest neighbors queries, and propose three different algorithms of query execution. An extensive experimental study on real-life data sets explores the performance characteristics of the proposed algorithms by showing their advantages and disadvantages.

Key words: Content–Addressable Network, Peer–to–Peer, Metric Space, Similarity Search, Nearest Neighbor Search

1 Introduction

Traditionally, search has been applied to structured (attribute-type) data yielding records that exactly match the query. A more modern type of search, similarity search, is used in content-based retrieval for queries involving complex data types such as images, videos, time series, text and DNA sequences.

⋆ This work was partially supported VICE project (Virtual Communities for Education), funded by the Italian government, and by DELOS NoE, funded by the European Commission under FP6 (Sixth Framework Programme).

* Corresponding author.

Email addresses: fabrizio.falchi@isti.cnr.it (Fabrizio Falchi), claudio.gennaro@isti.cnr.it (Claudio Gennaro), zezula@fi.muni.cz (Pavel Zezula).

Similarity search is based on gradual rather than exact relevance using a distance metric that together with the database forms a mathematical *metric space*. The obvious advantage of similarity search is that the results can be ranked according to their estimated relevance. However, experience with the current, mostly centralized, similarity search structures reveals linear scalability with respect to the data search size, which is not acceptable for the expected dimension of the problem.

Peer–to–Peer (P2P) architectures seem to solve the problem of scalability, and several scalable and distributed search structures have been proposed even for the most generic case of metric space searching (see Section 2 for a survey). They mostly concentrate on the *similarity range* queries where the execution algorithms satisfying (1) the autonomy of updates and (2) no central coordination or flooding strategies, are easier to implement. Since no bottleneck occurs, the structures are scalable and high performance is achieved through parallel query execution on individual peers (computer nodes).

Since the number of closest objects is typically easier to specify than the search range, users prefer *nearest neighbors* queries. For example, given an image, it is easier to ask for 10 most similar ones according to an image proximity criterion than to define the similarity threshold (i.e., the range), quantified as a real number. However, nearest neighbors algorithms are much more difficult to implement in P2P environments. The main reason is that traditional (optimum) approaches are based on a *priority queue* with a ranking criterion, which sequentially decides the order of accessed data buckets. In fact, an existence of centralized entities and sequential processing are completely in contradiction with decentralization and parallelism objectives of any P2P search network.

Capitalizing on our previous work of similarity range search through MCAN (Falchi et al., 2005), in this article we propose and experimentally test several nearest neighbor search algorithms. We first summarize the necessary background in Section 2, including the related work. Then in Section 3 we define the main properties of the MCAN. Section 4 describes alternative strategies for the nearest neighbor search, while the results of experimental testing are reported in Section 5. The paper concludes in Section 6.

2 Background

The most fundamental research results to our proposal are the Content-Addressable Network (CAN) (Ratnasamy et al., 2001a) as the storage infrastructure and the metric space concept as an abstraction of nearness (Chávez et al., 2001). In the following, we provide the necessary background and survey relevant

literature.

2.1 Content–Addressable Network (CAN)

The CAN is a distributed hash table that uses a function for mapping "keys" onto "values" defining positions of keys in the table. In the CAN, the table is represented by a finite set of peers. Each peer of the network is dynamically associated with a partition of an N-dimensional Cartesian space. Usually, the Cartesian space is an N-torus (i.e., the coordinate space wraps), and is targeted to store (X, V) pairs, where X is the "key" and V is the "value" associated with X. Assuming the "key" as a representation of content, basic operations of the CAN are insertion, lookup and deletion of respective (X, V) pairs. Formally, if we refer the domain of X as $\mathcal{D}$, we can define the mapping function G of the CAN as follows:

$$G : \mathcal{D} \rightarrow R^N, \tag{1}$$

where R^N is an hyper–rectangular region of $\mathbb{R}^N$.

The principle of the CAN is to divide the hyper-rectangular region R^N in a finite number of distinct rectangular zones, each of them associated with exactly one peer of the network. The peers are responsible for storing and searching of objects covered by their zones. Moreover, each peer is aware of the peers that cover adjacent zones, i.e., its neighbors.

Given a "key", the lookup function returns coordinates of the zone into which the key belongs. This is useful for insertion, deletion, and retrieval purposes. The search starts from an arbitrary peer of the CAN structure and proceeds by routing a message towards its destination by using a simple greedy forwarding to the neighbor with coordinates closest to the destination zone. In general, if we divide the R^N uniformly in h zones, each peer maintains $2N$ neighbors. Furthermore, the average routing path length is given by $(N/4)h^{(1/N)}$.

To simplify the discussion in the rest of the paper, we consider any element (key) of $\mathcal{D}$ as object, neglecting the fact that there is always a value V associated with it.

2.2 Metric Spaces

The mathematical metric space is a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the *domain* of objects and d is the *distance function* $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ able to compute distances between any pair of objects from $\mathcal{D}$. It is typically assumed that

the smaller the distance, the closer or more *similar* the objects are. For any distinct triple of objects $X, Y, Z \in \mathcal{D}$, the distance must satisfy the following properties:

$$\begin{aligned} d(X,X) &= 0 && \textit{reflexivity} \\ d(X,Y) &> 0 && \textit{strict positiveness} \\ d(X,Y) &= d(Y,X) && \textit{symmetry} \\ d(X,Y) &\leq d(X,Z) + d(Z,Y) && \textit{triangle inequality} \end{aligned}$$

Let $\mathcal{F} \subseteq \mathcal{D}$ be the data-set. There are two basic types of similarity queries. The **range** query retrieves all objects which have a distance from the query object $Q \in \mathcal{D}$ at most the specified threshold (range or radius) r:

$$\{\forall X \in \mathcal{F} \mid d(X,Q) \leq r\}.$$

The **nearest neighbor** query returns the object that is the nearest (having the shortest distance) to the query object Q. We can extend this type of query to return k nearest objects that form a set $\mathcal{K} \subseteq \mathcal{F}$ such that $|\mathcal{K}| = k$ and:

$$\forall X \in \mathcal{K}, Y \in (\mathcal{F} - \mathcal{K}) : d(Q,X) \leq d(Q,Y).$$

In (Falchi et al., 2005), we proposed and analyzed a distributed similarity search structure supporting execution of range queries. In this article, we concentrate on the more natural but also more difficult form of similarity queries, that is the nearest neighbors queries.

2.3 Pivot Mapping and Filtering

In this section, we discuss general pivoting strategies proposed in MCAN for mapping metric space objects into vectors and for filtering undesirable objects during search operations.

2.3.1 Mapping

In general, the pivot-based algorithms can be viewed as a mapping F from the original metric space $\mathcal{M} = (\mathcal{D}, d)$ to an N-dimensional vector space. The mapping assumes a set $\{P_1, P_2, \ldots P_N\}$ of objects from $\mathcal{D}$, called pivots, and for each database object $X \in \mathcal{K}$, the mapping determines its characteristic (feature) vector as:

$$F(X) : \mathcal{D} \rightarrow \mathbb{R}^N \ = (d(X,P_1), d(X,P_2), \ldots d(X,P_N)) \qquad (2)$$

We obtain a new metric space as $\mathcal{M}_N(\mathbb{R}^N, d^\infty)$. Using the original objects, the distance between objects in this new space can be evaluated as:

$$d^\infty(F(X), F(Y)) = max_i \, |d(X, P_i) - d(Y, P_i)| \tag{3}$$

Since the triangle inequality for d in the original metric space $\mathcal{M}$ holds

$$\forall i \; d(X, Y) \geq |d(X, P_i) - d(Y, P_i)| \,, \tag{4}$$

we have

$$d(X, Y) \geq \max_i |d(X, P_i) - d(Y, P_i)| = d^\infty(F(X), F(Y)). \tag{5}$$

For any pair of objects, the distance in the derived space $\mathcal{M}_N$ is never larger than the distance in the original metric space $\mathcal{M}$. In this way, the mapping is *contractive*, i.e., d^∞ in $\mathcal{M}_N$ is a lower bound of the original distance d in $\mathcal{M}$.

2.3.2 Filtering

At the search time, we compute for a query object Q the query feature vector $F(Q) = (d(Q, P_1), d(Q, P_2), \ldots d(Q, P_N))$. Performing a range query with radius r, we can avoid the evaluation of $d(X, Q)$ for objects X that satisfy

$$d^\infty(F(X), F(Q)) > r. \tag{6}$$

During the evaluation of k nearest neighbors (kNN) with the temporary result $X_1, \ldots, X_k$, we can avoid the evaluation of $d(Q, X)$ if

$$d^\infty(F(X), F(Q)) > d(X_k, Q). \tag{7}$$

In other words, the object X can be discarded if there exists a pivot P_i such that

$$| \, d(Q, P_i) - d(X, P_i) \, | > d(X_k, Q). \tag{8}$$

For more details about the use of pivots in metric spaces, see for example (Bustos et al., 2001).

2.4 Related Work

Many metric-based indexing principles and index structures have been proposed, focusing on the pruning of search space at query time, and several comprehensive surveys describe individual approaches (Hjaltason and Samet, 2003; Chávez et al., 2001; Zezula et al., 2006). Most of these indexes are static

or main memory structures and, therefore, not very suitable for large volumes of data.

The most significant dynamic and disk-oriented structures are the M-Tree (Ciaccia et al., 1997) and the D-Index (Dohnal et al., 2003). However, even with these structures, the similarity search becomes too expensive when the stored data volume grows, because the search costs increase linearly with respect to the size of the dataset (Dohnal et al., 2003). This fact calls for an attempt to exploit a distributed processing.

Restricting to multi-dimensional range and kNN queries in vector spaces, several recent works have proposed distributed structures for such tasks. Unfortunately, these structures are often designed for specific applications (for example spatial data) typically using vectors of low dimensionality. The vector space approach cannot be applied on many important datasets where similarities are measured by functions such as the Hausdorff distance, Jaccard's coefficient, edit distance, etc.

The MAAN structure (Cai et al., 2003) extends the Chord protocol to support multi-attribute and range queries by means of uniform locality preserving hashing. This system expects the exact knowledge of the attribute domain distributions. Moreover, no experiments on real-life datasets are provided in the paper. The structure is also not applicable to general metric data and has not defined any other similarity queries accept for the range search.

Prasanna, Yang and Garcia-Molina (Ganesan et al., 2004) show in the SCRAP structure a way to adapt kd-trees, which support multi-dimensional range queries, by exploiting the Chord protocol. Unfortunately, this approach becomes inefficient for more than two dimensions. In the same paper, the authors propose the MURK structure that uses the *space-filling curves* together with CAN (improved by skip pointers). Although no real-life dataset experiments are given, this structure is probably efficient for low-dimensional range queries in terms of routing costs and number of query-relevant nodes. The efficiency was not verified on high-dimensional data and cannot be applied to data that is metric by its nature.

The Mercury (Bharambe et al., 2004) provides a protocol for routing multi-attribute range-based queries. The efficiency of this protocol is measured in terms of number of hops and total number of messages for routing of three-dimensional range queries. Again, the motivation and potential application of this protocol is less general and slightly different from that addressed by this paper.

The Skip Graphs (Aspnes and Shah, 2003) and the SkipNet structure (Harvey et al., 2003) (both improved by (Aspnes et al., 2004)) extend the concept of Distributed Hash Tables by the principles of locality and load balancing. These

features give the opportunity to support range queries on one dimension. These approaches do not aim on more complex similarity queries.

The pSearch approach (Tang et al., 2003) uses two traditional information retrieval algorithms – *vector space model* and *latent semantic indexing* – to build a decentralized P2P information retrieval system based on the CAN routing protocol. This concept, defining similarity between a document and a query by means of their common terms, is only suitable for the text retrieval.

Tanin, Nayar and Samet (Tanin et al., 2005a) introduce a P2P generalization of a *quadtree* index. The authors propose range and nearest neighbor algorithms over this structure (Tanin et al., 2005b). Application of this structure is limited to the *spatial domain*.

Banaei-Kashani and Shahabi (Banaei-Kashani and Shahabi, 2004) formalize the problem of vector-based similarity search in P2P Data Networks and propose the SWAM – a family of small-world based access methods. This concept provides a general solution for the range and nearest neighbors search in the vector-based datasets.

Four different distributed structures have been proposed for indexing and similarity search in the metric data. The first two, the GHT* (Batko et al., 2005) and the VPT*, are native metric index structures whereas the other two, our MCAN (Falchi et al., 2005) and the M-Chord (Novák and Zezula, 2006), transform the metric search issue into a different problem and take advantage of some well-known solutions. Assuming similarity range queries, the pros and cons of these four structures are deeply analyzed in (Batko et al., 2006) on several real-life datasets.

3 Principles of MCAN

The basic idea of our approach is to extend the CAN architecture so that it can manage objects $\mathcal{F}$ of a generic metric space $\mathcal{M} = (\mathcal{D}, d)$. However, in metric spaces it is not possible to exploit any knowledge of coordinate information, and only distances between objects can be computed. To cope with this problem, we use the pivots paradigm for mapping the objects of the metric space to an N dimensional vector space. In particular, let $P_1, \ldots, P_N$ be the pivots selected from the metric data-set, we map an object $X \in \mathcal{D}$, by means of the function $F()$ (introduced in the Section 2.2) defined by Eq. 2.

This virtual coordinate space is used to store the object X in the MCAN structure, specifically in the peer that owns the zone in which the point $F(X)$ belongs. Note that, the coordinate space of the MCAN is Cartesian but the

distance between two objects is evaluated by means of the d^∞ distance instead of the canonical Euclidean distance. Routing in MCAN works in the same manner as for the original CAN structure. An MCAN peer maintains a coordinate routing table that holds the IP address and virtual coordinate zones of each of its immediate neighbors in the coordinate space.

3.1 Notation

In this section, we provide necessary definitions needed for a clear presentation of our results. We use the capital letter for indicating metric space objects $X \in \mathcal{D}$, the overline small letter for denoting the corresponding vector in the coordinate space $\overline{x} \in \mathbb{R}^N$ ($\overline{x} = F(X)$), and x_i for representing the values of its i-th coordinate. As we have already explained, the mapping in MCAN is contractive, therefore $d^\infty(\overline{x}, \overline{y}) \leq d(X, Y)$ always holds.

We denote a peer of MCAN by the bold symbol $\mathbf{n}$. Each peer $\mathbf{n}$ maintains its region information referred as $\mathbf{n}.R$. Moreover, since the region $\mathbf{n}.R$ is an hyper–rectangle it can be uniquely identified by its vertex closest to the origin, denoted as $\mathbf{n}.R.\overline{v} = (\mathbf{n}.R.v_1, \mathbf{n}.R.v_2, \ldots, \mathbf{n}.R.v_N)$, and by the lengths of the relative sides, i.e., $\mathbf{n}.R.l_1, \mathbf{n}.R.l_2, \ldots, \mathbf{n}.R.l_N$. More precisely, the region $\mathbf{n}.R$ is defined as follows

$$\mathbf{n}.R = \{\forall \overline{x} \in \mathbb{R}^N \mid \forall\ i,\ \mathbf{n}.v_i \leq x_i < \mathbf{n}.v_i + \mathbf{n}.l_i\}$$

The peer $\mathbf{n}$ also maintains the set of the neighbor peers' information $\mathbf{n}.M \subset \{\mathbf{n}_1, \ldots, \mathbf{n}_h\}$.

Given an object $Q \in \mathcal{D}$ and a range r, we define $\langle \overline{q}, r \rangle$ as the hypercube in $\mathbb{R}^N$ with center $\overline{q} = F(Q)$ and side $2r$.

We can now introduce the formal definition of an N-dimensional MCAN structure, referred as MCAN^N, which is composed of a set of h ($h > 0$) network peers $\{\mathbf{n}_1, \ldots, \mathbf{n}_h\}$ such as:

(1) $\forall\ i, j \mid i \neq j \quad \mathbf{n}_i.R \cap \mathbf{n}_j.R = \emptyset$
(2) $\bigcup_{i=1}^{h} \mathbf{n}_i.R = R^N$
(3) $\mathbf{n}_i \in \mathbf{n}_j.M \Leftrightarrow$
$\exists k \mid (\mathbf{n}_i.R.v_k + \mathbf{n}_i.R.l_k = \mathbf{n}_j.R.v_k) \vee (\mathbf{n}_j.R.v_k + \mathbf{n}_j.R.l_k = \mathbf{n}_i.R.v_k)$,
$\forall w \neq k\ [\mathbf{n}_i.R.v_w, \mathbf{n}_i.R.v_w + \mathbf{n}_i.R.l_w[\cap [\mathbf{n}_j.R.v_w, \mathbf{n}_j.R.v_w + \mathbf{n}_j.R.l_w[\neq \emptyset$

In the definition, Point 1. states that the zones covered by the network peers do not overlap. Point 2. states that the union of the zones cover the whole MCAN^N space R^N (there are no holes). Finally, Point 3. declares the condition for a network peer $\mathbf{n}_i$ to be a neighbor of $\mathbf{n}_j$ (as explained in Section 2).

3.2 Construction

An important feature of the CAN structure is its capability to dynamically adapt to data-set size changes. As we will see in the experimental evaluation, we are interested in preserving the scalability of the MCAN, which means that we want to maintain a stable the response time of query execution. Since the number of objects a peer can maintain is limited, when a peer exceeds its limit it splits by sending a subset of its objects to a free peer that takes responsibility for a part of the original region. Note that, limiting the number of objects each peer can maintain, we also limit (reduce) the number of distance computations a peer have to compute during a query evaluation.

It is important to observe that in some cases we might want to use all the peers available in the network. Previous work like (Ratnasamy et al., 2001a) have studied this possibility in a generic CAN structure by allowing a peer to split even if it does not exceed its storage capacity. Obviously, such methodology can also be applied in our MCAN. On the other hand, in a P2P environment, we would like to let the peers the possibility to freely join and leave the network, without affecting its consistency. As explained in (Ratnasamy et al., 2001a), this is possible with a CAN, which even provides some fault-tolerance capabilities (Saia et al., 2002).

Since pivots need be determined before the insertion starts, we assume a characteristic subset of the indexed dataset (about 5000 objects) is known at the beginning. For selecting the pivots, we use the Incremental Selection algorithm described in (Bustos et al., 2001). This algorithm tries to maximize the average distance d^∞ between two arbitrary objects in the derived space (i.e., $d^\infty(F(X), F(Y))$).

3.3 Insertion

An insert operation can be initiated in any peer of the MCAN. It starts by mapping the inserted object X to the virtual coordinate space using function $F()$, and proceeds by checking if $\overline{x} = F(X)$ lies in the zone maintained by the peer $\mathbf{n}$, i.e., $\overline{x} \in \mathbf{n}.R$. If this is not the case, the peer forwards the insertion request. From this point, the insertion proceeds with the greedy routing algorithm used in standard CAN structures: the inserting peer forwards the insertion operation to the neighbor peer which is closer to the point $\overline{x}$ by using the d^∞ distance. The objective is to find the peer $\mathbf{n}$ for which $\overline{x} \in \mathbf{n}.R$, minimizing the number of messages. We refer to this special peer as $\mu(\overline{q})$ (i.e., $\overline{q} \in \mu(\overline{q}).R$). If $\overline{x}$ lies in the region maintained by the receiving peer, the object X is stored there, otherwise a neighbor peer is selected with the same

technique and the insert operation is forwarded again until the object X is inserted.

The peer $\mu(\overline{x})$, which stores the object X must reply to the peer that started the insert operation. If the peer $\mu(\overline{x})$ exceeds its capacity it splits. Eventually, the object X is inserted in $\mu(\overline{x})$ or in the new allocated peer.

3.4 Split

In MCAN, we apply a balanced split, i.e., the resulting regions contain the same number of objects. During this process, the splitting peer just requests a peer from a free peer list to join the network, and one half of the metric objects is then reallocated there.

If we define $\mathbf{n}_1$ as the splitting peer, $\mathbf{n}_1.R$ as the old region, $\mathbf{n}_1.R'$ as the new one, and $\mathbf{n}_2$ as the new peer, the split regions must satisfy the following three equations:

$$\mathbf{n}_1.R' \cup \mathbf{n}_2.R = \mathbf{n}_1.R \; ; \quad \mathbf{n}_1.R' \cap \mathbf{n}_2.R = \emptyset$$

Moreover, to respect these constrains, we create the new two regions by dividing the original one along one coordinate of the space. Therefore, the new regions, $\mathbf{n}_1.R'$ and $\mathbf{n}_2.R$, must satisfy the following equations:

$$\mathbf{n}_1.R'.v_s = \mathbf{n}_1.R.v_s;\ \mathbf{n}_2.R.v_s = \mathbf{n}_1.R'.v_s + \mathbf{n}_1.R'.l_s;\ \ \mathbf{n}_2.R.l_s = \mathbf{n}_1.R.l_s - \mathbf{n}_1.R'.l_s$$

Note that that we only have to choose s and $\mathbf{n}_1.R'.l_s$. In order to decide s, for each dimension i we find $\mathbf{n}_1.R'.l_i$ that divide the objects into two halves. Moreover, we choose to split along the dimension that maximizes the length of the shortest side.

After the splitting process, the peer $\mathbf{n}_1$ sends a message to all its neighbors $\mathbf{n}_1.M$ informing them about the update of its region. It also sends information about the new peer to the neighbors that are also neighbors of $\mathbf{n}_2$. The new peer is informed by $\mathbf{n}_1$ about its neighbors $\mathbf{n}_2.M$ (note that $\mathbf{n}_2.M \subseteq \mathbf{n}_1.M$). At the end, $\mathbf{n}_1$ can discard information about the peers that are not more its neighbors.

4 Searching for Nearest Neighbors

Whenever we want to search for similar objects using the range search, we must specify the maximal distance of objects that qualify. However, it can be very difficult to specify the radius without some knowledge about the data and the used metric space. For example, the range $r = 3$ of the edit distance metric

represents less than four edit operations between the two strings, which has a clear semantic meaning. However, a distance of two color-histogram vectors of images is a real number, which cannot be so easily quantified. When a too small query radius is specified, the result set can even be empty and a new search with a larger radius is needed to get a result. On the other hand, queries with too large query radii might be computationally expensive, and the result sets might contain many not significant objects.

An alternative way to search for similar objects is to use the nearest neighbors queries. Such queries guarantee the retrieval of k most relevant objects, i.e., the set of k objects with the shortest distances to the query object Q. Though the problem of executing k nearest neighbors queries is not new and many algorithms have been proposed in the literature, see for example (Hjaltason and Samet, 1999) for many references and additional readings, the distributed kNN query processing have not been systematically studied.

4.1 kNN Search in MCAN

In MCAN, we have developed three different strategies to perform kNN queries: Parallel Execution (PE), Sequential Execution (SE), and Mixed Mode Execution (MME). Each of these techniques has its advantages and disadvantages which will be discussed later.

All these three strategies start by locating the peer that contains the query object Q. We refer to this special peer as $\mu(\overline{q})$. The location of $\mu(\overline{q})$ is performed exactly the same way as for the insertion operation described in Section 3.3. The kNN proceeds in the peer $\mu(\overline{q})$ and finds the k objects nearest to Q. Note that, we assume that there are at least k objects in $\mu(\overline{q})$. Because of the splitting rule this condition is guaranteed for any k less or equal to half of the peer capacity. However, in case k is greater than the number of objects contained in $\mu(\overline{q})$, the algorithms could be easily modified by forwarding the kNN request to the most promising peer until the temporary result list contains k objects. Therefore, the first k objects, i.e., the objects with the shortest distances to Q, are the candidates for the kNN result. However, there may be other objects in different peers' regions that are closer to the query than some of those k candidates. Nevertheless, since the MCAN space is contractive, these objects are within the distance to the k-th objects found to the query. In order to verify if there are other peers involved in the query, $\mu(\overline{q})$ controls if the hypercube $\langle \overline{q}, d(X_k, Q) \rangle$ is completely contained in $\mu(\overline{q}).R$ (where X_k is the k-th element of the candidate result set of the kNN). If this is true, the kNN search correctly terminates and the k objects retrieved by $\mu(\overline{q})$ represent the result of the kNN query.

When $\langle \overline{q}, d(X_k, Q) \rangle$ is not completely contained in the region $\mu(\overline{q}).R$, we have to check if there are other peers that maintain objects near to Q, respecting the k objects found in the peer $\mu(\overline{q})$. From this stage, our three proposed algorithms start working differently. In particular, the algorithms differ in the way they propagate the kNN query execution among the involved peers. The generic behavior of these three approaches can be characterized as follows:

PE All the peers overlapping the hypercube $\langle \overline{q}, d(X_k, Q) \rangle$ are involved in the kNN operation. The overlapping peers that receive the query first forward the kNN query and only then they start evaluating the query on their local data.

SE The $\mu(\overline{q})$ peer only involves the most near neighbor to Q. This peer first computes locally the kNN query updating its temporary result list and only after this involves the next peer most near to Q, is needed.

MME The $\mu(\overline{q})$ peer involves its neighbors that overlap the hypercube $\langle \overline{q}, d(X_k, Q) \rangle$. Every peer first computes locally the kNN query updating its temporary results list and only after this involves its neighbors that overlap the updated hypercube $\langle \overline{q}, d(X_k, Q) \rangle$.

The query propagation of PE requires an application level multicast. In fact, starting from $\mu(\overline{q})$, the query is forwarded to all peers which overlap the hypercube $\langle \overline{q}, d(X_k, Q) \rangle$. The multicast algorithm proposed in (Ratnasamy et al., 2001b) with improvement and corrections described in (Jones et al., 2002) are used in MCAN in order to reduce the number of replicated messages. The same algorithm is also used by MME. Actually, in MME the query propagates as in the PE except for the fact that the hypercube can reduce its size during the query propagation.

It is important to note that the three algorithms differ not only in the temporal sequence in which the peers are involved but they also differ in the number of accessed peers. In fact, the algorithms MME and SE can take advantage of the partial kNN evaluation for optimizing the query by possibly reducing the number of peers which the kNN query must be forwarded to. This optimization cannot be exploited in PE and it is optimum for the SE algorithm. On the other hand, while the parallelization of the kNN operation is maximum for the PE, for SE there is no parallelization at all. The third approach (MME) represents a trade off between PE and SE strategies.

The three algorithms also differ in terms of the total number of distance computations. In fact, during the kNN query forwarding, in all the three approaches, the peers send along with k and the query object Q a list of the distances of the current candidate result set of nearest neighbors. More precisely, a peer $\mathbf{n}$ which evaluates the kNN updates the ordered list L^k defined as:

$$L^k(i) = d(X_i, Q),$$

where the object X_i belongs to the merged result set of the kNN query evaluated both by **n** and by a certain number of peers (it depends on the algorithm) which have been involved.

Concerning the PE approach, L^k is sent by the peer $\mu(\overline{q})$ to all its neighbors involved in the kNN query (if any). This information is then forwarded (unmodified) to the other peers involved in the kNN query. L^k together with the pivot objects can be exploited by a peer in order to reduce the number of distance computations during the kNN evaluation exploiting Eq. 8. It is clear that in PE approach L^k cannot be updated because the peers first forward the query to their neighbors and then they proceed with the query evaluation. On the contrary, in the MME approach the peers can produce and forward a more accurate version of L^k with the advantage of being able to reduce both the number of peers involved and the number of distance computations with respect to PE. In fact, the number of peers involved in the query and the number of distance computations both depend on the distance $d(X_k, Q)$ which typically decrease as the kNN computation proceeds. Finally, the query evaluation before the forwarding in MME reduces the degree of parallelism.

The SE algorithm takes the maximum advantage of information stored in L^k. The peers are involved one after the other according to their region distance from Q, and each peer evaluates the query before forwarding it to the next peer. For this reason, the peer that receives the forward of the kNN must know the current list of the peers that could be involved in the query. This list consists of the set of the not yet involved peers whose distances (of their regions) from Q are less than or equal to $d(X_k, Q)$ and that are neighbors of previously involved peers. This is necessary since the neighbor of a peer that is involved in the kNN needs not be a neighbor of the next peer involved in the SE sequence. Note that at the end of the kNN computations (performed by each peer) this list is pruned by removing the peers whose distances from Q become greater than the actual value of $d(X_k, Q)$. When the list is empty, the operation terminates and the result is sent to the requesting peer.

In general, we can consider two aspects of the kNN operation costs: its parallelism, which is necessary for the scalability, and its total computational cost. The PE approach tries to maximize parallelism while the SE tries to minimize the total computational costs. MME is somewhere in the middle.

In Figures 1, 2, and 3, we sketch the algorithms of the approaches PE, MME, and SE, respectively. As can be seen in the sketches, MCAN does not make use of a coordinating peer. Any peer sends its result set to the requesting peer (i.e., the peer which started the kNN operation). The requesting peer merges the result lists coming from the involved peers. Note that, in the algorithms we assume the distances between the results and the query are sent together with the objects id, even if not reported in the algorithms. In Figure 4, we report

```
receive Q, L^k
d_k:=L^k(k); # d_k is d(X_k, Q)
N:={∀m ∈ n.M | ⟨q̄, d_k⟩ ⋂ m ≠ ∅};
if L^k = ∅ then
   # the peer is μ(q̄):
   (L^k, A) := searchkNN_local(Q, L^k);
   for each m ∈ N
      send Q, L^k to m;
   end for each
else
   for each m ∈ N
       send L^k to m;
   end for each
   (L^k, A) := searchkNN_local(Q, L^k);
end if
send A to the requesting peer
```

Figure 1. PE algorithm

```
receive Q, L^k
(L^k, A) := searchkNN_local(Q, L^k);
d_k:=L^k(k); # d_k is d(X_k, Q)
N:={∀m ∈ n.M | ⟨q̄, d_k⟩ ⋂ m ≠ ∅};
for each m ∈ N
   send Q, L^k to m;
end if
send A to the requesting peer
```

Figure 2. MME algorithm

the *searchkNN_local* used by the previous algorithms. In (Falchi et al., 2005), we described how the requesting peer realizes when the range query operation has terminated. Essentially, we maintain the list of the involved peers, which is forwarded during the query propagation. Each peer sends this information to the requester together with its result set. For this reason, any involved peer must answer to the requester even if its result set is empty.

The application level multicast algorithm is not reported in the algorithm sketches. Summarizing, PE and MME make use of this algorithm trying to reduce multiple forwarding of the same request to the same peer. What happens is that the forward to a peer is performed only if a complicated set of rules is satisfied. Please see (Jones et al., 2002) for more details.

```
receive Q, L^k, N
(L^k, A) := searchkNN_local(Q, L^k);
N := N + n.M; # adds the neighbors to the list
p :=GetNearestPeer(N, Q);
# select peers that will not be involved
T :=GetPeersFartherAwayThan(N, Q, d_k);
N := N − T − {p};
send Q, L^k, N to p;
send A to the requesting peer
```

Figure 3. SE algorithm

```
function (L^k, A) =searchkNN_local(Q, L^k)
note: n is the current peer and P_1, ..., P_w the pivots of the MCAN
   T^k := 0 # is a temporary list of objects and distances
   for i from 1 to k
      # for results belonging to other peers we just have distances
      T^k(i) := (null, L^k(i));
   end for
   for each X ∈ n # where n is the current peer
      # all these distances were pre-evaluated
      if | d(Q, P_i) − d(X, P_i) | ≤ d(L^k(k), Q) then
         # d(X, Q) is not pre-evaluated
         if d(X, Q) < d(L^k(k), Q) then
            T^k := T^k − {T^k(k)} + {X, d(X, Q)}; # preserving order
         end if
      end if
   end for each
   A := ∅;
   for i from 1 to k
      L^k(i) := T^k(i).distance; # for next peers we need only distances
      A := A + T^k(i).object; # object can be null
   end for
end function
```

Figure 4. Implementation of the function searchkNN_local

5 Performance Evaluation

5.1 Datasets and Measurements

For the experiments, the systems consisted of up to about 300 active peers (depending on the dataset). The peers had storage capacity of 5,000 objects. The implementations built up overlay structures over a high-speed LAN communicating via the TCP and UDP protocols. Note that, we are not talking

about simulation but about a real prototype.

It is important to observe that in order to evaluate the scalability performance of MCAN, in this experimental evaluation we maintain the list of available inactive peers and employ them during the split of an overloading peer. We are aware of the fact that maintaining the list of the inactive peers is quite unusual in a real P2P scenario; however, this approach was adopted just to study the scalability of MCAN with respect to a growing dataset, assuming that the greater the dataset the more peers are employed. The objective was to demonstrate that keeping the average number of objects per peer limited as the dataset grows, the response time of the system remains bounded, i.e., the structure scales well. These experiments were performed by inserting objects in the network in a random order, resulting in a growing number of peers due to the splitting rule described in Section 2.3.

To conduct this experimental evaluation, we selected the following significantly different real-life datasets:

VEC 45-dimensional *vectors* of extracted color image features. The similarity function d for comparing the vectors is a *quadratic-form distance* (Seidl and Kriegel, 1997). The distribution of the dataset is quite uniform and such a high-dimensional dataspace is extremely sparse.

TTL titles and subtitles of Czech books and periodicals collected from several academic libraries. These *strings* are of lengths from 3 to 200 characters and are compared by the *edit distance* (Levenshtein, 1965) on the level of individual characters. The distance distribution of this dataset is skewed.

DNA protein symbol *sequences* of length sixteen. The sequences are compared by a *weighted edit distance* according to the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970). This distance function has a very limited domain of possible values – the returned values are integers between 0 and 100.

Observe that none of these datasets can be efficiently indexed and searched by a standard vector data structure. In the reported experiments, we used three pivots for building MCAN structure (i.e., MCAN^3) and 16 pivots for filtering. In (Falchi et al., 2005), we tried several different dimensions of the MCAN space and we found that $N = 3$ is an optimal solution. For more details about how the number of pivots affects the performance of the filtering please refer to (Bustos et al., 2001).

All the presented performance characteristics of query processing have been taken as an average over 100 queries by randomly choosing query objects not belonging to the dataset.

It is important to remark that, in a real scenario as the one we are evaluating, the calculation of the distance function d has typically a high computational

cost. Therefore, the main objective of a metric-based data structures is to reduce the number of distance computations at query time. The number of distance computations is typically considered an indicator of the structure efficiency. In practice, we assume that the costs of other operations are negligible compared to the distance evaluation time.

Concerning the distributed environment, we use the following two characteristics to measure the computational costs of a query:

- *total distance computations* – the sum of the number of distance computations on all employed peers,
- *parallel distance computations* – the maximal number of distance computations performed in a sequential manner during the parallel query processing.

Another indicator that we monitored is the *percentage of peers* (with respect to the total number) that were involved by the query processing and the *number of candidate results*.

In order to better interpret the performance figures of the three kNN algorithms presented above, we compare the results of the experiments with an ideal kNN algorithm, designated RQ, which is equivalent to a single range query. RQ works as follows: once we have obtained the result set of the kNN (evaluated using one of the three algorithms), we run a range query with radius $d(X_k, Q)$. The performance figures of RQ can be considered as the lower bounds (optimal) for the other kNN algorithms.

5.2 Number of peers involved in query execution

Figure 5 shows the average percentage of involved peers during the evaluation of kNN for increasing values of k and for the entire dataset. The performance figures as function of k are only reported for the VEC dataset (1 million objects and 260 peers), since the results of the other two datasets are very similar.

From these experiments, we can see that SE is not only the best algorithm in terms of number of involved peers but it is also optimum. In fact, its results are the same as those we obtained with RQ. PE involves much more peers and MME is not far from PE. Note also that, the number of peers grows almost linearly with k. Moreover, we can observe that for bigger k MME tends to be slightly better than PE. In fact, the more are the peers involved, the more is the relevance of intermediate results updated during the forwarding of the operation.

Figures 6, 7, and 8 show the average percentage of involved peers as the dataset grows. For all algorithms, the percentage of involved peers decreases with the

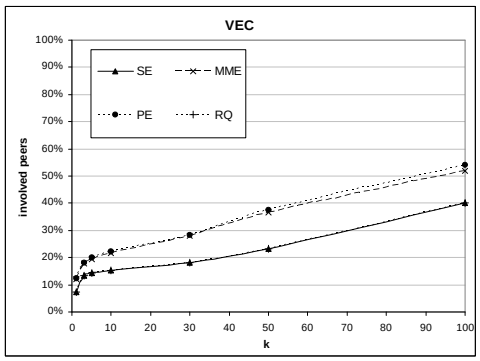

Figure 5. Percentage of involved peers for various k for VEC dataset.

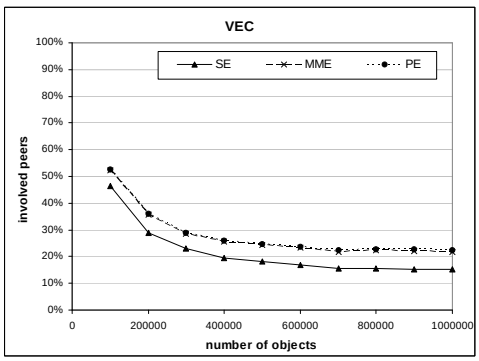

Figure 6. Percentage of involved peers for growing VEC dataset ($k = 10$).

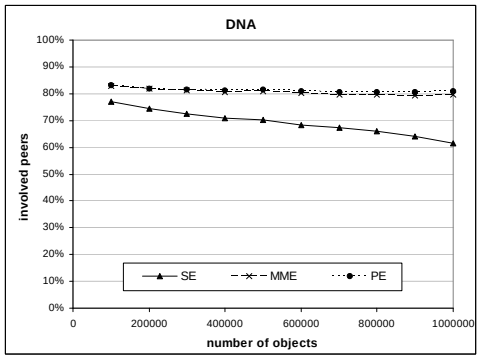

Figure 7. Percentage of involved peers for growing DNA dataset ($k = 10$).

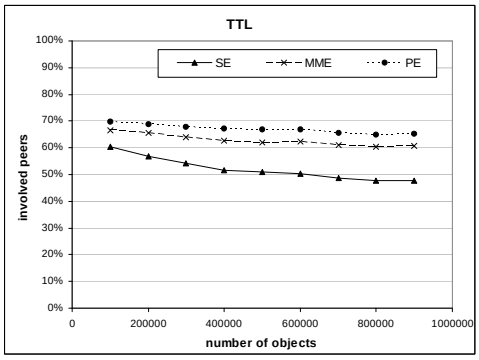

Figure 8. Percentage of involved peers for growing TTL dataset ($k = 10$).

number of objects of the dataset. This can be explained by the fact that in general as the dataset grows the distance of the k-th object to the query Q decreases. Furthermore, as the number of peers increases, the average volume of zones maintained by the peers decreases.

Regarding the differences between the three datasets' results, we can see that the DNA dataset is much more difficult to index than the VEC one. The most important reason is that the DNA metric function has a very limited number of discrete distance values. The TTL dataset is in the middle but not far from the DNA dataset. In fact, the TTL and DNA distance functions are not much different even if the object are, in their meaning, completely different.

5.3 Total number of distance computations

In Figure 9, we report the total number of distance computations during the kNN operation for the entire dataset and various k. Even though not optimum, SE is very near to the results obtained with RQ. In fact, it was also the algorithm that involved less peers. The price of this good result is the serialization of the operation which we will study more in details later on. Regarding

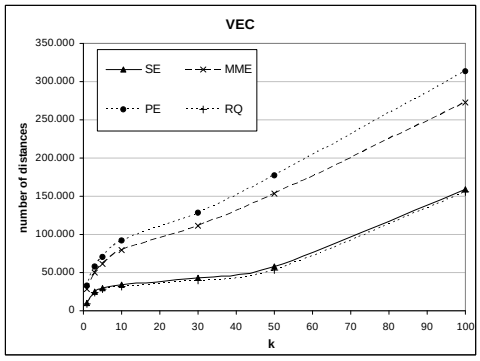

Figure 9. Total number of distance computations for various k for VEC dataset.

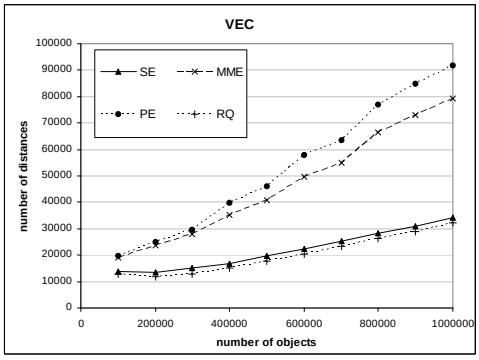

Figure 10. Total number of distance computations for growing VEC dataset (K=10).

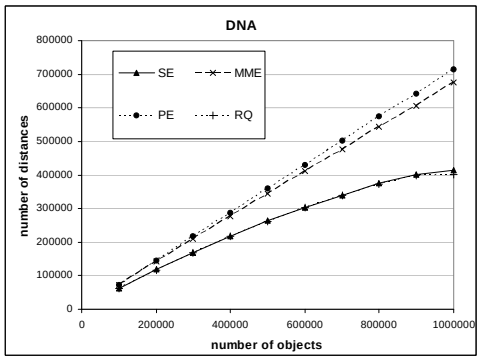

Figure 11. Total number of distance computations for growing DNA dataset (K=10).

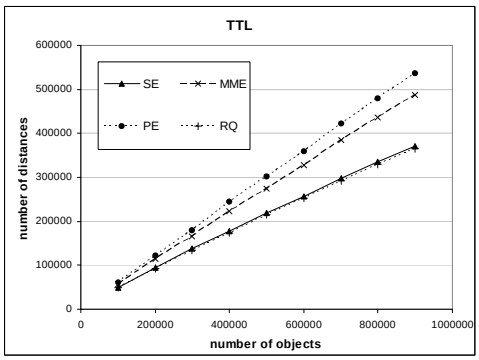

Figure 12. Total number of distance computations for growing TTL dataset (K=10).

MME, we can see that performing kNN computations before forwarding the query significantly reduces the total number of distance computations.

Figures 10, 11, and 12 show the total number of distance computations as the dataset size grows. Note that, in all the algorithms the total number of distance computations grows when we increase the dataset size. However, the most important property that we should expect from a P2P data structure such as the MCAN is its scalability of the search operations, which is achieved through parallelism.

5.4 Parallel cost of kNN

In Figure 13, we report the parallel distance computations for increasing values of k. As explained above, this performance figure can be considered as the parallel cost of the operation. From this experiment, it becomes clear what is the price the SE algorithm has to pay to obtain better results in terms of percentage of involved peers and total number of distance computations

– the cost of the operation grows quickly with k. Furthermore, from Figures 14, 15, and 16 we can see that the parallel cost of SE grows with the dataset size, which means that the algorithm does not scale well. On the contrary, PE scales well and it is very near to the optimum RQ. Note that, the parallel cost of RQ does not grow with the dataset size, because as the number of objects increases the corresponding range radius $d(X_k, Q)$ becomes smaller. Finally, MME, which gave better results than PE in terms of the number of involved peers and total number of distance computations, does not scale as well as the PE, but it is not very far from it.

Considering the parallel cost as the response time of the network it must be noticed that, because the results are sent by each involved peer directly to the requester, there are some preliminarily results before the kNN operation is completed. Regarding SE and MME, the order in which the peers are visited guarantees that good results will be available to the requester before the end of the operation. However, in the PE algorithm the results are supposing to arrive sooner and almost at the same time because of the parallelism (except for the $\mu(\overline{q})$ peer), although the use of preliminarily results by the requester can make MME and SE more appealing in some scenarios.

5.5 Candidate results

In all the algorithms, as the kNN evaluation proceeds, the peers send the partial results of their local kNN evaluation to the requesting peer $\mu(\overline{q})$ (which is the peer which started the kNN operation). We call these partial results *candidate results*.

Since the user needs an ordered list of k results, $\mu(\overline{q})$ performs also the task of merging, sorting, and pruning, if necessary, the results beyond the k-th received from the peers involved in the kNN. In Figure 17, we report the number of candidate results received for the three algorithms as the dataset grows. The SE algorithm is near the optimum. In fact, it is very near to RQ which, by definition, exactly retrieves k results (it can retrieve more results just in case there are more objects at the same distance of the k-th result). The PE algorithm is the worst and it does not scale well (except for the DNA and TTL datasets). On the contrary, MME is between the other two algorithms and its behavior seems growing sub-linearly.

Notice that, generally, the metric distance evaluation in the metric space is very expensive and the cost of the operations performed by the requester is then negligible. In fact, it does not have to evaluate any distance (we assumed that the distance between the objects and the query are sent together with the results). With these experiments we just wanted to show that there could

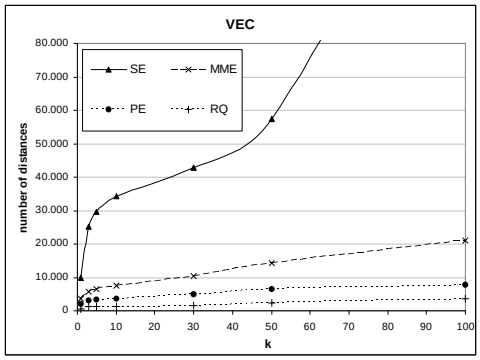

Figure 13. Parallel distance computations for various k for VEC dataset.

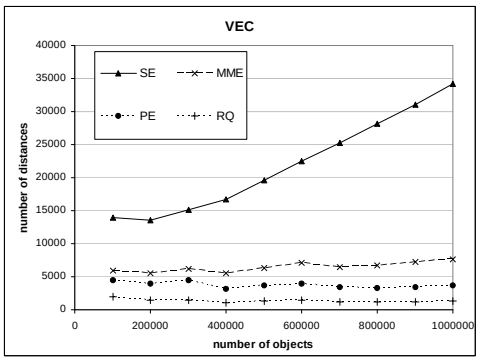

Figure 14. Parallel distance computations for growing VEC dataset (K=10).

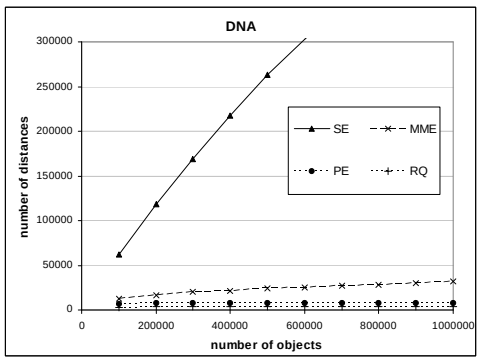

Figure 15. Parallel distance computations for growing DNA dataset (K=10).

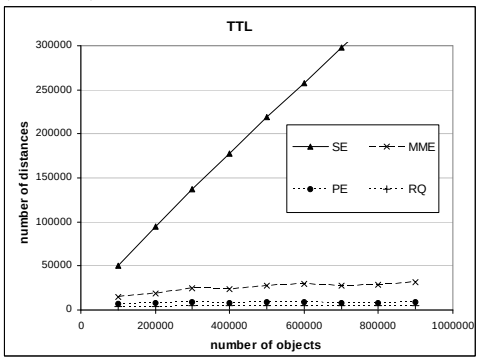

Figure 16. Parallel distance computations for growing TTL dataset (K=10).

be problems of scalability in terms of candidate results for the PE algorithm.

6 Conclusions and future work

Many P2P applications need processing complex data for which there is no ordering and only pair-wise distances (dissimilarities) can be decided by specific functions. We have considered the case where the dataset and the function form the metric space, so our approach offers a high extensibility – many different forms of data and queries can be processed with a single index structure. We have concentrated on a much needed and probably the most complex form of queries: the nearest neighbors queries.

We have proposed three strategies for such query execution, using the MCAN similarity search structure, originally developed for processing range queries. Extensive performance evaluation on real-life data processed by our experimental system reveals the following pros and cons of individual approaches.

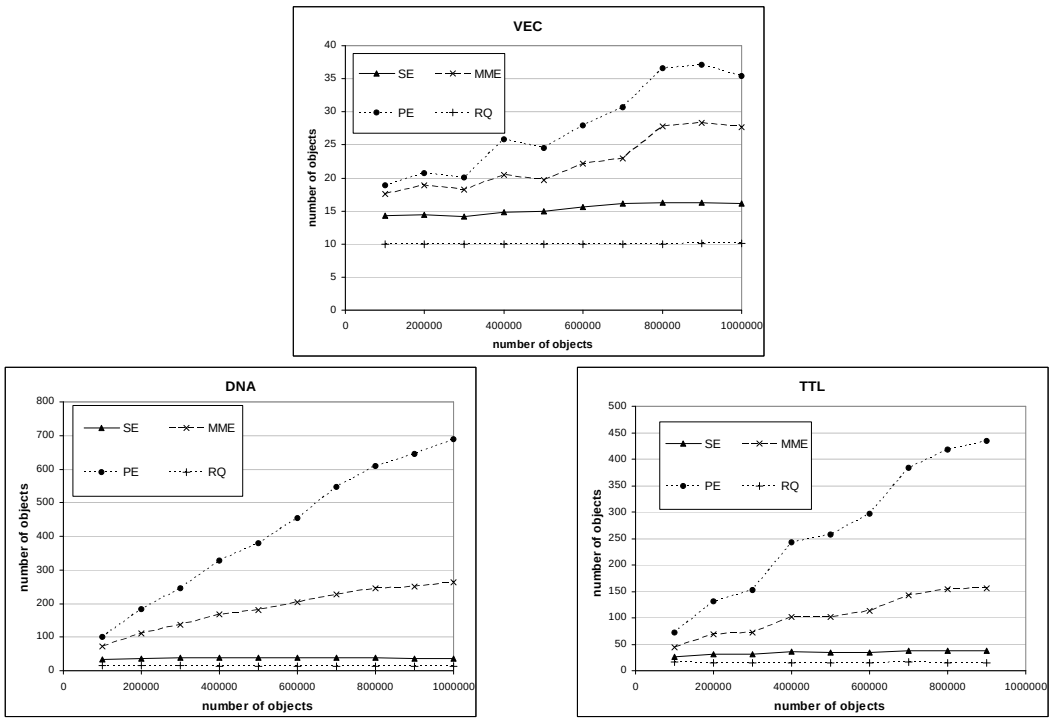

Figure 17. Average number of candidate results for growing dataset (K=10).

The experiments revealed that the SE approach is not suitable for scenarios where scalability of the response time is critical. In fact, as Figures 14, 15, and 16 demonstrate, the parallel distance computations grows quickly (and linearly) as the size of the dataset increases. Instead, MME and especially PE, exhibit a parallel distance computation much more bounded. Because of the fact that P2P architectures are generally used to solve the problem of scalability we think SE is, in most of the case, impracticable.

In terms of total computational cost (Figures 10, 11, and 12) MME performs better than PE, even if not so well as the SE. Consuming less resources MME should leave more space for parallelism between independent operations. To decide which is the best choice between MME and PE, we can also take into account the number of candidate results. As experiments of Figures 17 show, MME scale better than PE.

From these observations, we think that the MME approach is the best choice in general. It responds well to the demand of the scalability in terms of response time and consumes less resources than PE.

An interesting direction of investigation is to generalize the kNN algorithms by parameterizing their behavior. Let $\alpha, \beta \in [0, 1]$ be the parameters for this new algorithm and $\beta > \alpha$. A peer performing the kNN first involves its neighbors whose regions overlaps $\langle \overline{q}, \alpha d(X_k, Q) \rangle$. After the local execution of the kNN query the peer involves both the not yet involved neighbor that is closest to

the query $\overline{q}$ and those neighbors whose regions overlaps $\langle \overline{q}, \beta d(X_k, Q) \rangle$. Note that for SE $\alpha, \beta = 0$, for MME $\alpha = 0, \beta = 1$, for PE $\alpha = 1$ and β is useless because all the neighbors are involved before the local execution of the kNN.

Our future work will concentrate on inter–query parallelism and approximate range and nearest neighbors algorithms, which would trade some imprecision in search results with additional improvement of performance.

References

Aspnes, J., Kirsch, J., Krishnamurthy, A., 2004. Load balancing and locality in range-queriable data structures. In: PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing. ACM Press, New York, NY, USA, pp. 115–124.

Aspnes, J., Shah, G., 2003. Skip graphs. In: SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 384–393.

Banaei-Kashani, F., Shahabi, C., 2004. Swam: a family of access methods for similarity-search in peer-to-peer data networks. In: CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management. ACM Press, New York, NY, USA, pp. 304–313.

Batko, M., Falchi, F., Novák, D., Zezula, P., 2006. On scalability of the similarity search in the world of peers. In: INFOSCALE '06: Proceedings of the First International Conference on Scalable Information Systems. IEEE Computer Society.

Batko, M., Gennaro, C., Zezula, P., 2005. Similarity grid for searching in metric spaces. In: Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures, 6th Thematic Workshop of the EU Network of Excellence DELOS. Vol. 3664 of Lecture Notes in Computer Science. Springer, Berlin / Heidelberg, pp. 25–44.

Bharambe, A. R., Agrawal, M., Seshan, S., 2004. Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, NY, USA, pp. 353–366.

Bustos, B., Navarro, G., Chávez, E., 2001. Pivot selection techniques for proximity searching in metric spaces. In: SCCC'01: Proceedings of the XXI Conference of the Chilean Computer Science Society. pp. 33–40.

Cai, M., Frank, M., Chen, J., Szekely, P., 2003. Maan: A multi-attribute addressable network for grid information services. In: GRID '03: Proceedings of the Fourth International Workshop on Grid Computing. IEEE Computer Society, p. 184.

Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J. L., 2001. Searching in metric spaces. ACM Comput. Surv. 33 (3), 273–321.

Ciaccia, P., Patella, M., Zezula, P., 1997. M-tree: An efficient access method for similarity search in metric spaces. In: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases. Morgan Kaufmann, pp. 426–435.

Dohnal, V., Gennaro, C., Savino, P., Zezula, P., 2003. D-index: Distance searching index for metric data sets. Multimedia Tools Appl. 21 (1), 9–33.

Falchi, F., Gennaro, C., Zezula, P., August 2005. A content-addressable network for similarity search in metric spaces. In: DBISP2P '05: Proceedings of the the 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing, Trondheim, Norway. pp. 126–137.

Ganesan, P., Yang, B., Garcia-Molina, H., 2004. One torus to rule them all: multi-dimensional queries in P2P systems. In: WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases. ACM Press, New York, NY, USA, pp. 19–24.

Harvey, N., Jones, M. B., Saroiu, S., Theimer, M., Wolman, A., March 2003. Skipnet: A scalable overlay network with practical locality properties. In: USITS '03: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems. Seattle, WA.

Hjaltason, G. R., Samet, H., 1999. Distance browsing in spatial databases. ACM Trans. Database Syst. 24 (2), 265–318.

Hjaltason, G. R., Samet, H., 2003. Index-driven similarity search in metric spaces. ACM Trans. Database Syst. 28 (4), 517–580.

Jones, M. B., Theimer, M., Wang, H., Wolman, A., December 2002. Unexpected complexity: Experiences tuning and extending can. Tech. Rep. MSR-TR-2002-118, Microsoft Research.

Levenshtein, V. I., 1965. Binary codes capable of correcting spurious insertions and deletions of ones. Problems of Information Transmission 1, 8–17.

Needleman, S. B., Wunsch, C. D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal Of Molecular Biology 48, 443–453.

Novák, D., Zezula, P., May 2006. M-chord: A scalable distributed similarity search structure. In: INFOSCALE '06: Proceedings of the First International Conference on Scalable Information Systems. IEEE Computer Society.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S., 2001a. A scalable content-addressable network. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, NY, USA, pp. 161–172.

Ratnasamy, S., Handley, M., Karp, R. M., Shenker, S., 2001b. Application-level multicast using content-addressable networks. In: NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication. Springer-Verlag, pp. 14–29.

Saia, J., Fiat, A., Gribble, S. D., Karlin, A. R., Saroiu, S., 2002. Dynamically

fault-tolerant content addressable networks. In: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems. Springer-Verlag, London, UK, pp. 270–279.

Seidl, T., Kriegel, H.-P., 1997. Efficient user-adaptable similarity search in large multimedia databases. In: The VLDB Journal. pp. 506–515.

Tang, C., Xu, Z., Dwarkadas, S., 2003. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, NY, USA, pp. 175–186.

Tanin, E., Harwood, A., Samet, H., 2005a. A distributed quadtree index for peer-to-peer settings. In: ICDE '05: Proceedings of the International Conference on Data Engineering. IEEE Computer Society, pp. 254–255.

Tanin, E., Nayar, D., Samet, H., 2005b. An efficient nearest neighbor algorithm for P2P settings. In: Proceedings of the 2005 national conference on Digital government research. Digital Government Research Center, pp. 21–28.

Zezula, P., Amato, G., Dohnal, V., Batko, M., 2006. Similarity Search: The Metric Space Approach. Vol. 32 of Advances in Database Systems. Springer-Verlag.