Approximate Similarity Search from another "Perspective" (EXTENDED ABSTRACT) *

Giuseppe Amato and Pasquale Savino

ISTI-CNR, Pisa, Italy
firstname.lastname@isti.cnr.it

1 Introduction

We propose a new approach to perform approximate similarity search in metric spaces [8]. The idea at the basis of this technique is that when two objects are very close one to each other they 'see' the world around them in the same way. Accordingly, we can use a measure of dissimilarity between the view of the world, from the perspective of the two objects, in place of the distance function of the underlying metric space. To exploit this idea we represent each object of a dataset by the ordering of a number of reference objects of the metric space according to their distance from the object itself. In order to compare two objects of the dataset we compare the two corresponding orderings of the reference objects. We show that efficient and effective approximate similarity searching can be obtained by using inverted files, relying on this idea. We also show that the proposed approach performs better than other approaches proposed in literature.

2 Perspective based space transformation

Let $\mathcal{D}$ be a domain of objects and $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$ be a metric distance function between objects of $\mathcal{D}$. Let $RO \subset \mathcal{D}$, be a set of reference objects chosen from $\mathcal{D}$. Given an object $o \in \mathcal{D}$, we represent it as the ordering of the reference objects RO according to their distance d from o. More formally, an object $o \in \mathcal{D}$ is represented with $\bar{o} = O^{RO}_{d,o}$, where $O^{RO}_{d,o}$ is the ordered list containing all objects of RO, ordered according to their distance d from o.

We denote the position in $O^{RO}_{d,o}$ of a reference object $ro_i \in RO$ as $O^{RO}_{d,o}(ro_i)$. For example, if $O^{RO}_{d,o}(ro_i) = 3, ro_i$ is the 3rd nearest object to o among those in RO. We call $\bar{\mathcal{D}}$ the domain of the transformed objects. $\forall o \in \mathcal{D}, \bar{o} \in \bar{\mathcal{D}}$. Figure 1 exemplifies the transformation process. Figure 1a) sketches a number of reference objects (black points), data objects (white points), and a query object (gray point). Figure 1b) shows the encoding of the data objects in the transformed space. We will use this as a running example throughout the reminder of the paper.

As we anticipated before, we assume that if two objects are very close one to each other, they have a similar view of the space. This means that also the orderings of the

* This work was partially supported by the DELOS NoE and the Multimatch project, funded by the European Commission under FP6 (Sixth Framework Programme).

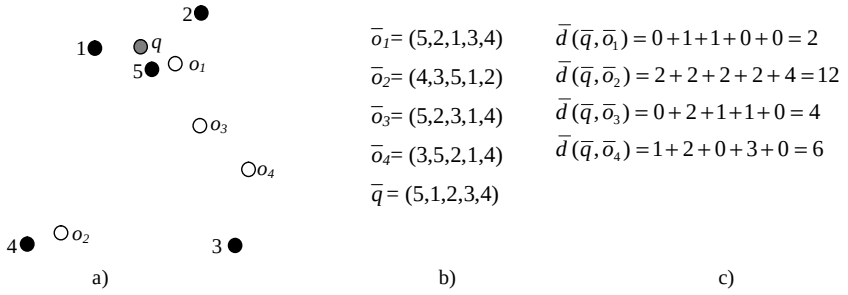

Fig. 1. Example of perspective based space transformation. a) Black points are reference objects; white points are data objects; the gray point is a query. b) Encoding of the data objects in the transformed space. c) Distances in the transformed space

reference objects according to their distance from the two objects should be similar. In order to measure the similarity between two orderings we use the *Spearman Footrule Distance* (SFD) [5], which is a popular measure to compare ordered lists. Given two ordered lists S_1 and S_2, containing all objects of RO, the SFD between S_1 and S_2 is computed as the sum of the absolute differences between positions of objects in the two orderings. More formally

$$SFD(S_1, S_2) = \sum_{ro \in RO} |S_1(ro) - S_2(ro)|$$

We denote the distance between objects in the transformed domain as $\bar{d}(\bar{o_1}, \bar{o_2}) = SFD(O^{RO}_{d,o_1}, O^{RO}_{d,o_2})$.

The transformed domain $\bar{\mathcal{D}}$ and the distance $\bar{d}$ can be used to perform approximate similarity search in place of the domain $\mathcal{D}$ and the distance function d. Figure 1c) shows the distance, computed in the transformed space, of the data objects from the query object. It can easily be seen that it is consistent (it gives the same ordering) with the actual distance of data objects from the query.

3 Using inverted files

Let us suppose that we have a dataset $X \subset \mathcal{D}$ and a query $q \in \mathcal{D}$. Suppose we want to search for the k objects of X nearest to q. An exhaustive approach is that of ordering the entire dataset X according to the distance from q and to select the first k objects. Let $\bar{X}$ be the dataset in the transformed space and $\bar{q} \in \bar{\mathcal{D}}$ the transformed query. The approximate ordering of X with respect to q can be obtained in $\bar{\mathcal{D}}$ by computing the distance $\bar{d}(\bar{q}, \bar{o}), \forall o \in X$. In the following we will show that this ordering can be obtained by representing (indexing) the transformed objects with inverted files and using search algorithms derived from the full text search area [6, 7]. Let us see this in details.

We can index transformed objects with inverted files as follows. Entries (the lexicon) of the inverted file are the objects of RO. The posting list associated with an entry

IN: query: q,
reference objects: RO,
posting lists associated with reference objects;
OUT: The set of accumulators: A
1. Set $A \leftarrow \{\}$.
2. For each $ro \in RO$
3. Let pl be the posting list associated with ro
4. For each $(o, O^{RO}_{d,o}(ro)) \in pl$
5. If $a_o \notin A$
6. Set $a_o = 0$
7. Set $A \leftarrow A \cup \{a_o\}$
8. Set $a_o = a_o + |O^{RO}_{d,q}(ro) - O^{RO}_{d,o}(ro)|$

Fig. 2. Basic searching algorithm using inverted files.

$ro \in RO$ is a list of pairs $(o, O^{RO}_{o}(ro)), o \in X$, that is a list where each object o of the dataset X is associated with the position of the reference object ro in $\bar{o}$. In other words, each reference object is associated with a list of pairs each referring an object of the dataset and the position of the reference object in the transformed object's representation. For instance, an entry $(o, 7)$ in the posting list associated with reference object ro, indicates that ro is the 7th closest object to o among those in RO.

Therefore, the inverted file has the following overall structure:

$$
\begin{aligned}
& ro_1 \rightarrow ((o_1, O^{RO}_{o_1}(ro_1)), \ldots, (o_n, O^{RO}_{o_n}(ro_1))) \\
& \ldots \\
& ro_m \rightarrow ((o_1, O^{RO}_{o_1}(ro_m)), \ldots, (o_n, O^{RO}_{o_n}(ro_m)))
\end{aligned}
$$

where n is the size of the dataset X and m is the size of the set of reference objects RO.

A basic algorithm that quickly computes the distance $\bar{d}$ of all objects of the dataset X from q using an inverted file data structure is given in Figure 2. The Algorithm uses an accumulator a_o, associated with each object o found, to incrementally compute the distance $\bar{d}(\bar{o}, \bar{q})$. The set of accumulators A is initially empty (line 1.). The posting lists of the various entries are accessed (lines 2. and 3.) and for each entry in a posting list (line 4.) if the object is seen for the first time a new accumulator is added to the list of accumulators (lines 5.–7.). Then the value of the accumulator is updated by adding the difference in position of the current reference object ro, in $O^{RO}_{d,q}$ and $O^{RO}_{d,o}$(line 8.). At the end of the algorithm execution all objects are associated with an accumulator that contains their distance $\bar{d}$ from the query object. It is easy to maintain the accumulators ordered during the algorithm execution so that at the end we have the entire dataset ordered. This algorithm is very similar to algorithms that incrementally compute the dot product in text retrieval systems. The difference is that here we compute the Spearman Footrule Distance.

In the reminder of the article we will modify this basic idea to gain orders of magnitude in performance. We will proceed according to these guidelines: 1) we will reduce

the search cost by reducing the number of posting lists accessed during search (the approach proposed above accesses all posting lists); 2) we will reduce the size of the inverted file by reducing the number of posting lists where each object is referred (at the moment each object is referred by every posting list); 3) we will further reduce the search cost by reducing the amount of entries read in every posting list (at the moment all entries of a posting list are read).

4 Searching with the k_s closest reference points

In order to improve the performance of the above described approach, we can simply represent the query objects with a subset of RO. More specifically, when we want to process a query object q, instead of representing it as the ordering $O^{RO}_{d,q}$ of all the objects in RO, we rather represent it as the ordering of the k_s ($k_s \leq \#RO$) reference objects of RO closest to q. We denote this ordering as $\bar{q} = O^{RO}_{k_s,d,q}$. The distance $\bar{d}$ is now computed by using a variant of the Spearman Footrule Distance, the *Induced Footrule Distance* (IFD), defined as:

$$IFD(S_1, S_2) = \sum_{ro \in S_1} |S_1(ro) - S_2(ro)|$$

In order to support this optimization, the search algorithm given in Figure 2 needs only to be modified in line 2. In fact, it should not access the posting lists associated with all reference objects. Rather it only accesses the posting lists associated with the k_s closest reference objects to q. Line 2. is therefore changed to "For each $ro \in O^{RO}_{k_s,d,q}$".

5 Indexing with the k_i closest reference points

The idea of taking just the closest reference objects can also be used to represent any object that has to be indexed, rather than just the query. Let $k_i \leq \#RO$ be the number of reference objects used for indexing. In this case every object can be represented as $\bar{o} = O^{RO}_{k_i,d,o}$, using a smaller amount of reference objects. Note that, in this case, different objects will be typically represented by different reference objects, given that different objects will have different neighbor reference objects. This representation of an object will be clearly smaller than using all reference objects. In addition, this has also the effect of reducing the size of the inverted file. In fact every object will be just inserted into k_i posting lists, by reducing their size and by also reducing the search cost.

The IFD computes the distance by summing the position differences of reference objects in the query and data object encoding. However, now it can happen that a reference object occurs in the query encoding and it does not occur in an object's encoding. When this happens we suppose that the position difference of the missing reference objects is greater than the maximum possible. That is, we suppose that the difference is $k_i + 1$.

Let us now discuss how the search algorithm has to be used with this modification to the indexing strategy. The basic algorithm that we gave previously in Figure 2, computes the distance of objects from the query incrementally using the accumulators. This techniques relies on the fact that every object appears exactly once in every

IN: query: q,
reference objects: RO,
posting lists associated with reference objects,
number of reference objects used for indexing: k_i,
number of reference objects used for searching: k_s;
OUT: The set of accumulators: A
1. Set $A \leftarrow \{\}$.
2. For each $ro \in O^{RO}_{k_s,d,q}$
3. Let pl be the posting list associated with ro
4. For each $(o, O^{RO}_{d,o}(ro)) \in pl$
5. If $a_o \notin A$
6. Set $a_o = (k_i + 1) * k_s$
7. Set $A \leftarrow A \cup \{a_o\}$
8. Set $a_o = a_o - (k_i + 1) + |O^{RO}_{d,q}(ro) - O^{RO}_{d,o}(ro)|$

Fig. 3. Searching algorithm that uses an inverted file where the closest k_i reference objects were used to index objects.

posting list. Therefore every posting list contributes to compute the distance of every object. However, as discussed above, some reference objects might not occur in some object's encoding and consequently some objects might be missing from some posting lists. In fact, when an object is indexed with its k_i closest reference objects, it will appear just in k_i posting lists. This means that when processing a query, an object might be encountered in some posting lists and it might not be seen in others. This leads to miscalculation of the distance. In fact, when an object is not seen in a posting list, it is as if 0 is erroneously summed to its accumulator, in correspondence of the reference object associated with the posting list. However, note that 0 means that the position of the reference object in an object is exactly the same than its position in the query, while the actual position difference (so the distance) should be higher.

To solve this, we change the way in which the accumulators are used. Figure 3 shows the modified algorithm. When objects are seen for the first time the associated accumulator is initialized with the maximum possible distance for an object, which is $(k_i + 1) \cdot k_s$ (line 6.). In fact an object can be seen at most in k_s posting lists and we assume an unseen reference object having position $(k_i + 1)$, as discussed before. Every time an object is encountered its current distance is reduced by replacing (subtracting) the maximum possible difference, which is $(k_i + 1)$, with the actual position difference from the query (line 8.).

6 Accessing a fraction of the posting lists

Previous algorithms read the entire content of the accessed posting lists. However, also here we can find some opportunity for reducing the search cost. We can observe that in the posting lists there are some pairs that are more promising that others. Specifically, the most promising pairs are those whose position (the position of the reference object)

is closer to that of the query. In fact, the IFD is computed by summing the position difference of a reference object in the query and in the searched objects. Given that we are interested in the k closest objects to the query, it is likely that objects whose position difference is high will fall outside the first k. Therefore, we can use a threshold parameter MPD which indicates the Maximum Position Difference and we can access just the pairs whose position difference is below the threshold.

In order to do that efficiently we can maintain the pairs of the posting lists ordered according to their position and, rather than accessing the entire posting lists, we can sequentially scan just the portion that contain pairs whose position difference is below the threshold. For instance, suppose that reference object ro has position p in the query q, that is $O^{RO}_{d,q}(ro) = p$. Then, the search algorithm will just access the consecutive pairs of the posting list, associated with ro, whose position is in the interval $[p - MPD, p + MPD]$.

Note that the number of possible positions is much smaller than the number of pairs in a posting list. Therefore, sorting of the entries of the posting lists, according to the position, can be efficiently performed in linear time using a count-sort algorithm, after bulk insertion.

The revised ranking algorithm, which accesses a subset of the posting lists content, is given in Figure 4. It first computes the minimum (mp) and the maximum (Mp) position that read pairs should have (lines 3. and 4.). Then, it accesses just the pairs in that interval (line 5. 6.). Note that direct access to the initial position of the interval can be obtained by maintaining an index (an array) for each posting list that indicates the offset corresponding to possible positions. The index for a posting list has size equal to the maximum possible position, which is k_i, and can be stored at the beginning of the posting list itself. Note that the use of an index for the positions in the posting list makes it possible to store only the reference to the object, rather than the pair containing the object and its position. In fact the position can be inferred from the index, knowing the absolute position of the object in the posting list.

7 Comparisons

In this section we assess the performance of the proposed approach in contrast with other methods of approximate similarity search in metric spaces, proposed in literature. Comparisons among some relevant methods [9, 1, 3] were already presented in other works [1, 8]. Thus, for brevity we just compare the proposed technique with the method of approximate similarity search based on region proximity [1], which was previously recognized as the one offering the best performance.

Approximate similarity search based on region proximity uses the M-Tree [4] access method, and relies on a strategy of pruning non promising subtrees, during the search phase, measuring the probability of overlap of the underlying ball regions with the query region. Tradeoff between accuracy and search cost can be tuned using a threshold on this probability of overlap. In order to have an objective comparison we used a dataset of image visual features, as described in [2]. We also set to 4 KBytes the size of the disk block (the unit of disk access) in both cases.

IN: query: q,
reference objects: RO,
posting lists associated with reference objects,
number of reference objects used for indexing: k_i,
number of reference objects used for searching: k_s,
maximum allowed position difference: MPD;
OUT: The set of accumulators: A

1. Set $A \leftarrow \{\}$.
2. For each $ro \in O^{RO}_{k_s,d,q}$
3. Let $mp = max(0, O^{RO}_{k_s,d,q}(ro) - MPD)$,
$Mp = min(k_i, O^{RO}_{k_s,d,q}(ro) + MPD)$
4. Let pl be the posting list associated with ro
5. Let pl^{Mp}_{mp} the subset of pl containing pairs with positions between mp and Mp
6. For each $(o, O^{RO}_{d,o}(ro)) \in pl^{Mp}_{mP}$
7. If $a_o \notin A$
8. Set $a_o = k_i * k_s$
9. Set $A \leftarrow A \cup \{a_o\}$
10. Set $a_o = a_o - k_i + |O^{RO}_{d,q}(ro) - O^{RO}_{d,o}(ro)|$

Fig. 4. Search algorithm that accesses a fraction of the posting lists.

Results are reported in Figure 5. The graphs show the performance of the four variants of our proposal and the proximity based method when the number k of searched objects is 50. For each method we plotted the search cost, in terms of disk block reads, against the recall and the precision error (their description can be found in [8]). A logarithmic scale is used for the search cost axis.

Leaving out of consideration our baseline method, where all reference objects are used for indexing and searching, the remaining variants typically offer much higher accuracy at the same search cost with respect to the method based on proximity. Specifically, when the search cost is between 100 and 1000 disk accesses, our methods offers a recall around 0.5 and a position error of about 0.02. On the other hand, the proximity based method offers a recall around 0,1-0,2 and a position error above 0.03. Note that the point with highest accuracy (recall 1 and error 0), offered by the proximity based method, was obtained with extreme settings of the approximation threshold corresponding to execute in fact exact similarity search.

From the graphs it is also evident how the best variant of our method is when just a portion of the posting lists are accessed.

8 Conclusions

In this paper we presented an approach to approximate similarity search in metric spaces based on a space transformation that relies on the idea of perspective from a data point. We proved through extensive experimentation that the proposed approach has clear ad-

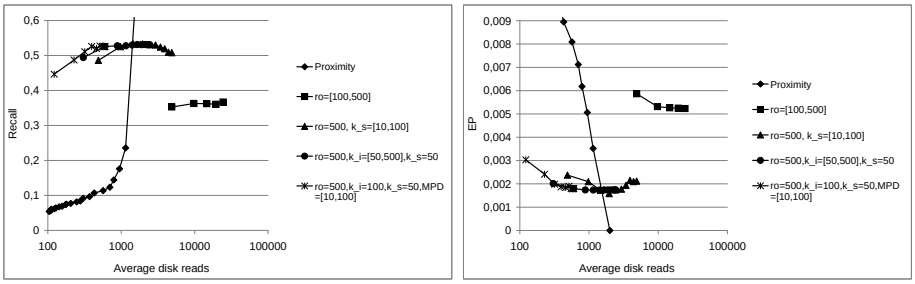

Fig. 5. Comparisons of various setting of the proposed approach against a state-of-the-art method. We consider the case when the number k of retrieved objects is 50. We measured the average recall and the position error in correspondence of the search cost, measured as number of disk clock reads.

vantages over other methods existing in literature. A major characteristics of the proposed technique is that it can be implemented by using inverted files, thus capitalizing on many years of investigation on efficient and scalable searching algorithms on this data structure. The proposed approach belongs to the category of general purpose access methods for similarity search. It can be applied to any application where the similarity search paradigm can be modelled using metric spaces.

References

1. G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, 21(2):192–227, 2003.
2. S. D. Bay. The uci kdd archive. Irvine, CA: University of California, Department of Information and Computer Science. http://kdd.ics.uci.edu.
3. P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *ICDE*, pages 244–255, 2000.
4. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 426–435. Morgan Kaufmann, 1997.
5. P. Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *IMS Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, Hawyard Ca, 1988.
6. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1983.
7. I. H. Witten, A. Moffat, and T. C. Bell. *Bell: Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.
8. P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.
9. P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with m-trees. *VLDB J.*, 7(4):275–293, 1998.