Experimental analysis of insertion costs in a naive dynamic MDF-tree

Luisa Micó and Jose Oncina

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante
P.O. box 99, E-03080 Alicante, Spain
{mico,oncina}@dlsi.ua.es
http://www.dlsi.ua.es

Abstract. Similarity search is a widely employed technique in Pattern Recognition. In order to speed up the search many indexing techniques have been proposed. However, the majority of the proposed techniques are static, that is, a fixed training set is used to build up the index. This characteristic becomes a major problem when these techniques are used in dynamic (interactive) systems. In these systems updating the training set is necessary to improve its performance. In this work, we explore the surprising efficiency of a naive algorithm that allows making incremental insertion in a previously known index: the MDF-tree.

1 Introduction

In the area of similarity search techniques, the metric space searching is an arising general approach that has received special attention. In this work we are interested in the most general case, when no assumption about the structure of the prototypes (points) is made. Some examples are protein sequences (represented by strings), skeleton of images (represented by trees or graphs), histograms of images, etc.

Actually, many techniques have been proposed based on this approach (some reviews can be found in [3][8][15]). These techniques, based on the use of any of the elementary types of similarity queries, have been used in many applications. For example, content based image retrieval [7], person detection or automatic image annotation [13], texture synthesis, image colourisation or super-resolution [1]. However, most of the existing techniques are static [14][2][10][11]. Then, the insertion or deletion of one object in the the index that has been built up in a preprocessing step, requires an complete rebuild that is very expensive.

In recent years, an increasing attention on interactive systems has been observed in the information technology society. In particular, during an interactive operation, new samples (new information) should be added in an on-line training process adapting the model to the present situation.

Some methods to speed up insertion/deletion points operations have been proposed. Some of them are focused in dynamic indexing in secondary memory ([4]) and others are focused to work in main memory [12]. In some cases, the

dynamic approaches have been proposed based as modifications of a previous existing static indexing algorithm but there are also techniques devised from the beginning as dynamic such as the M-tree [5].

In this paper we analyse experimentally some strategies to insert new samples in the *Most Distant from the Father tree* (MDF-tree). This type of structure have been used in several hierarchical search procedures ([9],[6]) but no dynamic approach has been proposed up to now for it.

2 New incremental indexing approximation

The MDF-tree is a binary indexing structure proposed and used in [9][6] based on a hyperplane partitioning approach. The main characteristic of the method is related to the selection of the representatives for the next partition. To build up the tree, firstly a pivot is randomly selected as the root of the tree (first level). After that, a new pivot, the most distant from the root, is selected and the space is divided according the distances to both pivots. On the following levels the space is recursively divided between the current pivot of the subspace represented by the node and the most distant pivot in the subspace. This procedure is repeated until each internal node has only one object.

For each node, the covering radius (the distance from the pivot to the most distant point in the subspace) is computed and stored in the respective node. This procedure is described in the algorithm 1.

The function build(ℓ,S) takes as arguments the future representative of the root node and the set of objects to include in the tree (excluding ℓ) and returns the MDF-tree that contains $S \cup \{\ell\}$. The first time that build(ℓ,S) is called, ℓ is a random object. In the algorithm M_T represents the representative of T, radius_T the covering radius, and T_L (T_R) the left (right) subtree of T.

Algorithm 1: **build(ℓ, S)**

Data:
  $S \cup \{\ell\}$: set of points to include in in T;
  ℓ: future left representative of T
create MDF-tree T
if *S is empty* **then**
  $M_T = \ell$
  $\text{radius}_T = 0$
  return T
end
$\text{radius}_T = \max_{x \in S} d(\ell, S)$
$r = \text{argmax}_{x \in S} d(\ell, S)$
$S_\ell = \{x \in S | d(\ell, x) < d(r, x)\}$
$S_r = \{x \in S | d(\ell, x) \geq d(r, x)\} - \{r\}$
$T_L = \text{build}(\ell, S_\ell)$
$T_R = \text{build}(r, S_r)$
return T

Algorithm 2: **insert**(T, x)

Data:
  T: MDF-tree
  x: object to insert in T
if $d(M_T, x) > r_T$ **then**
  return build($M_T, \{s | s \in T\} \cup \{x\} - \{M_T\}$)
end
if T_L *is empty* **then**
  return build($M_T, \{x\}$)
end
$d_\ell = d(M_{T_L}, x)$ // this distance computation can be avoided
$d_r = d(M_{T_R}, x)$
if $d_\ell < d_r$ **then**
  T_L = insert(T_L,x)
end
T_R = insert(T_R,x)
return T

Although several similarity search strategies can be applied, in this work we are focused in nearest neighbour search. The search procedure consists on a first-depth traversal through the tree, looking for the nearest neighbour. Given a node in the tree, the search continues through the child node whose representative is nearest to the query. Using different types of elimination rules, some branches of the tree are bounded and the search ends when no more possibilities to explore in the tree are possible. The last candidate is selected as the solution.

2.1 Rebuilding the tree

In this work we focus on a procedure to obtain incrementally the exact structure that will be obtained if a complete rebuild of the tree was made. Note that in such case the performance of the algorithm when searching is exactly the same as in the static case. Then no further research in search degradation performance is needed.

We are going to study a naïve approach. When a new object is inserted first we search the position where object should appear as pivot in a static built tree. Then, the affected part of the tree is completely rebuilt.

This may seems a quite expensive strategy, but as the pivots are very unusual objects (the farthest of its sibling pivot), big reconstructions of the tree happens with a very low probability compensating its high cost.

Let T be the MDF-tree built using a sample set S. Let x be the new object we are interested to include in the index, and let T' the MDF-tree built using the sample set $S \cup \{x\}$. The algorithm works rebuilding the subtree of T that is different from T'.

Let we denote by M_T the representative of the root node of the MDF-tree T, radius$_T$ be its covering radius, and T_L (T_R) the left (right) MDF-subtrees of T.

We have several cases:

- If $d(M_T, x) > r_T$, T' differs from T in the root node because the object x is selected in T' as the representative of the right node. Then the whole tree T is rebuilt in order to include x (see fig. 2).
- Otherwise, the roots of the trees T and T' are identical. Then we have two cases (see fig. **??**) :
  - if $d(M_{T_L}, x) < d(M_{T_R}, x)$ the object x should be inserted in the left tree T_L and then the tree T'_R is identical to T_R.
  - Conversely if $d(M_{T_L}, x) \geq d(M_{T_R}, x)$ the object should be inserted in T_R and the tree T'_L is identical to T_L.

Algorithm 2 shows the insertion procedure.

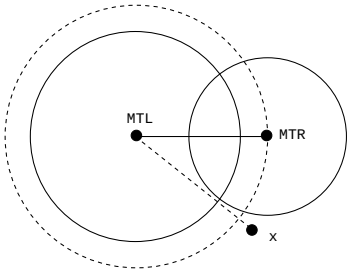

Fig. 1. Case when a complete rebuilt is needed

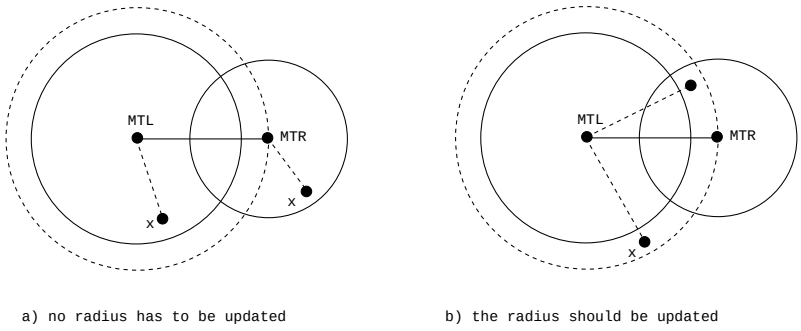

Fig. 2. Location of the inserted object

3 Experiments

Some experiments have been developed in order to study the performance of the new approach.

The experiments where performed by extracting 5, 10 and 15 dimensional points from the unit hypercube with a uniform distribution.

First in order to study the the distribution of the number of distance computations when an object is inserted, 10000 insertions of an object over a fixed MDF-tree with sizes 100, 1000, and 10000 was made. The number of distance computations were counted and its histogram was depicted in figure 3. We show only the case for dimension 10, the other cases are similar.

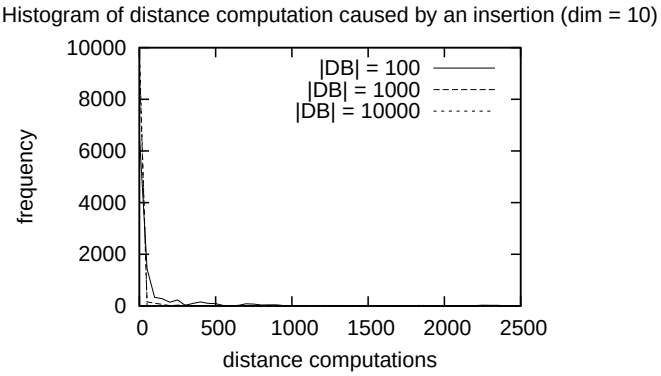

Fig. 3. Histogram of the distance computations caused by an insertion for 100, 1000, and 10000 database sizes in dimension 10

It can be observed that, as expected, almost all the insertions causes very few distance computations and, conversely, there are very few insertion that causes a big number of distance computations.

It can also be observed that this figure is very far of a normal distribution, then on the following experiments instead of using the variance we used the 95 percentile.

In the next experiment we have depicted the expected number of distance (over 1000 repetitions) for inserting a point when the databases grows from 100 to 10000 in steps of 100. The 95 percentile is also depicted (fig. 4). That means that although the expected number of distance computations is the line showed by the average in the 95% of the cases the number of distance computations is bellow the line of the 95 percentile.

Although these results are quite preliminary, it can be seen that the distance computations caused by an insertion seems to grow very slowly with the database size.

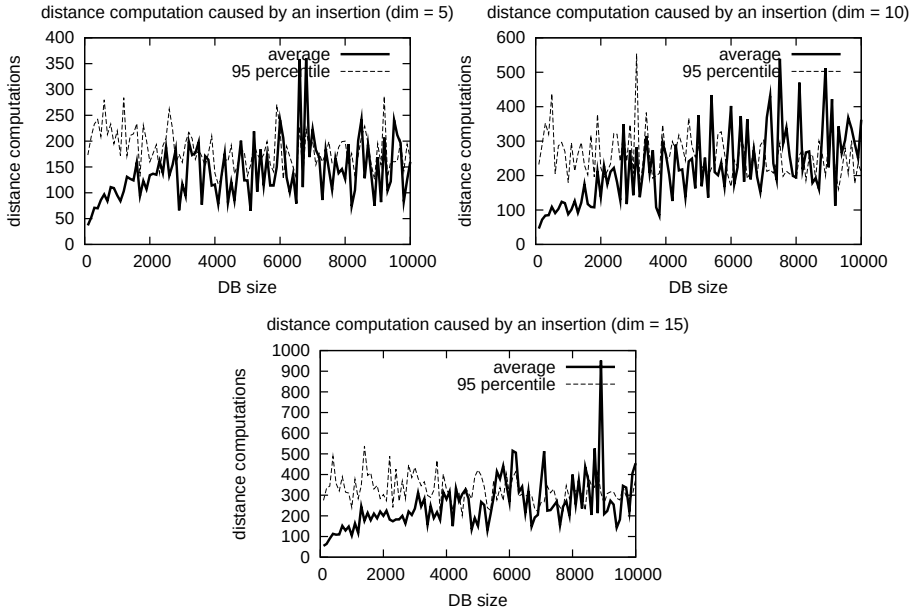

Fig. 4. Average distance computations caused by an insertion for increasing size database sets and for dimensions 5, 10, 15

4 Conclusions

We have proposed a naïve algorithm that allows the insertion in MDF-trees avoiding (with high probability) a complete rebuilt of the tree. We have also performed several artificial experiments to support this assertion.

In the future we plan to make more extensive experimentation and develop some ideas in order to have a theoretical support of this result.

We also plan to relax the condition on maintaining the same tree as if all the objects were inserted at once with the hope to decrease the number of distance computations and the stability of the algorithm.

5 Acknowledgements

This work has been supported in part by grants DPI2006-15542-C04-01 from the Spanish CICYT (Ministerio de Ciencia y Tecnología), the IST Programme of the European Community, under the Pascal Network of Excellence, IST-2002-506778, and the program CONSOLIDER INGENIO 2010 (CSD2007-00018).

References

1. S. Battiato, G. Di Blasi, and D. Reforgiato. Advanced indexing schema for imaging applications: three case studies. *Image Processing, IET*, 1(3):249–268, 2007.
2. S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the* 21^{st} *International Conference on Very Large Data Bases*, pages 574–584, 1995.
3. E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
4. Ada Wai chee Fu, Polly M. S. Chan, Yin ling Cheung, and Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *VLDB Journal*, 9:154–173, 2000.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on VLDB*, pages 426–435, Athens, Greece, 1997. Morgan Kaufmann Publishers.
6. Gómez-Ballester E., Micó L., and Oncina J. Some approaches to improve tree-based nearest neighbour search algorithms. *Pattern Recognition*, 39(2):171–179, February 2006.
7. G. Giacinto. A nearest-neighbor approach to relevance feedback in content based image retrieval. In *CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 456–463, New York, NY, USA, 2007. ACM.
8. G.R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.
9. L. Micó, J. Oncina, and R.C. Carrasco. A fast branch and bound nearest neighbor classifier in metric spaces. *Pattern Recognition Letters*, 17:731–73, 1996.
10. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
11. G. Navarro. Searching in metric spaces by spatial approximation. *VLDB Journal*, 11(1):28–46, 2002.
12. G. Navarro and N. Reyes. Dynamic spatial approximation trees. *J. Exp. Algorithmics*, 12:1–68, 2008.
13. A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.
14. P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.
15. P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer, 2006.