Dynamic Insertions in TLAESA fast NN Search Algorithm

Luisa Micó
Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante (SPAIN)
Email: mico@dlsi.ua.es

Jose Oncina
Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante (SPAIN)
Email: oncina@dlsi.ua.es

***Abstract*—Nearest Neighbour search (NNS) is a widely used technique in Pattern Recognition. In order to speed up the search many indexing techniques have been proposed. The need to work with large dynamic databases in interactive or online systems, has resulted in an increase interest in adapting or creating fast methods to update these indexes. TLAESA is a fast search algorithm that computes a very low number of distance computations with sublinear overhead using a branch and bound technique.**

In this paper, we propose a new fast updating method for the TLAESA index. The behaviour of this index has been analysed theoretical and experimentally. We have obtained a log-square upper bound of the rebuilding expected time. This bound has been verified experimentally on several synthetic and real data experiments.

Keywords*—*nearest neighbour; dynamic index; metric spaces; similarity search;

I. Introduction

Many pattern recognition applications implicitly use the concept of similarity search. This concept involves different types of search, being the nearest neighbour search the most often used.

A naive approach to nearest neighbour search involves a costly traversal of the whole database. To speed up this search, a wide range of techniques has been proposed in the literature [**?**].

In this work we are concerned with TLAESA fast nearest neighbour search algorithm [**?**]. As many other search algorithms, TLAESA builds, in a preprocessing step, a data structure (*index*) that is later used in search time to speed up the process. TLAESA surged as a combination of two previous search algorithms: LAESA [**?**] and MDF [**?**] to overcome some of their weaknesses: LAESA was very efficient in avoiding distance computations but has a linear average time complexity, and MDF has a sublinear time complexity but is not so efficient in reducing the number of distance computations. TLAESA combines the strengths of both algorithms: the average number of distance computations in a search shows to be independent of the database size and the average time complexity grows sublinearly with the database size.

Despite this features few attention has been devoted to the TLAESA index build complexity. In fact, this building time grows $O(n \log n)$ with the data base size. This feature makes it unusable in interactive systems where the full index should be rebuilt each time that a new data is inserted in the database. To overcome this problem, several proposals have been made for metric space indexes [**?**], [**?**], [**?**], [**?**].

In this work we follow an approach similar to the used in [**?**] and [**?**]. The idea is to follow the usual procedure that is used to build the index from scratch, but trying to reuse the existing index as possible. If we fall into a state where the preexisting index can not be reused, this part of the structure is deleted and completely rebuilt.

At the end of the process we obtain an index that is exactly the same that the one that we obtain if it were built from scratch. Then the efficiency of the search algorithm is not affected by the way the new object is inserted and no new experiments about the search efficiency are necessary.

The partial rebuildings of the index are particularly expensive. In this work we show that big rebuildings are very unusual and this low occurrence probability compensates its expensiveness. More precisely, we have found a square logarithmic upper bound to the average number of distance computations that is needed to rebuild the TLAESA index. Moreover, we have made some experiments with synthetic and real data to illustrate the validity of the bound.

II. The static index

Given a database D of n objects in a metric space E provided with a distance function $d(\cdot,\cdot)$, TLAESA builds two data structures:

- a table of distances
- a binary tree structure

A. Table of distances

The table of distances stores all the distances between a selected subset P (pivots) of objects in the database and all the objects in the database. The number $m \ll n$ of pivots to be selected is a parameter to be experimentally adjusted. Then the size of the table (mn) grows linearly with the database size.

There are several methods for selecting the pivots [**?**] [**?**]. Without loss of generality, in this work we apply the so called *maxmin*, a simple greedy method that has been used with excellent results.

Maxmin recursively selects the m pivots as follows:

- the first pivot (p_1) is randomly chosen. Let $P_1 = \{p_1\}$,
- the following pivots (p_i) are the farthest objects to the actual pivot set

$$p_i = \underset{r \in D}{\operatorname{argmax}} \min_{p \in P_{i-1}} d(p, r);\ P_i = P_{i-1} \cup \{p_i\}$$

B. Binary tree structure

This structure is based on the MDF-tree [**?**], built using an hyperplane partitioning approach.

The tree structure is just a way to represent a partition of the objects in the database. Each node t stores:

- an object (t_r) that acts as representative of the objects stored in its subnodes (the *partition*).
- the radius of the partition (t_ρ). That is, the distance from the representative to the most distant object in the partition.

As representative of the whole tree a random element of the database is chosen. Then, the rest of the tree is recursively built as follows:

- the representative (l_r) of the left subtree (l) is the same than the representative of his father (t_r);
- the representative (r_r) of the right subtree (r) is the most distant object to the left representative (l_r);
- the objects near to l_r than r_r goes to the left partition;
- the objects near to r_r than l_r goes to the right partition.

C. TLAESA search procedure

The TLAESA search procedure is similar to the MDF search procedure. The main difference is that, in order to avoid some distances computations, it uses a lower bound ($g(\cdot,\cdot)$) instead of the real distance.

$$d(x, q) \geq g(x, q) = \max_{p \in P} |d(p, q) - d(p, x)|$$

Note that the distances $d(p, x)$ are stored in the table. The distances $d(p, q)$ are computed and stored at the beginning of the procedure. Then, the computation of the lower bound can be done without computing any further distance.

The search of the nearest neighbour is a depth traversal of the tree. The algorithm recursively descents the tree following the nearest representative according to the lower bound. Once it reaches a leaf, a real distance is computed and the nearest neighbour is updated if necessary. Then it goes back in the recursivity avoiding descending by branches where it is impossible to find the nearest neighbour (for that, it uses the radius of the partition) [**?**].

III. The incremental index

The goal of the incremental algorithm is to be able to insert a new object q into the index recycling as much of the previous index as possible. The final aim of the dynamic algorithm is to obtain the exact index that would have been obtained if the index had been build from scratch, then, no search degradation performance occurs.

As the TLAESA index has two components we are going to study them separately.

A. Updating the table

We are going to follow the *minmax* algorithm that was used to build the initial table but now using the database $D \cup \{q\}$ [**?**].

The algorithm begins selecting the same random prototype that was used to build the previous index. Each time a new pivot is proposed two cases are possible:

- the proposed pivot is the object to be inserted (q), then the table from this point should be recomputed;
- the proposed pivot p is not q, then it should be the same that was chosen when the old index was built. In this case all the distances $d(p, \cdot)$ stored in the previous table can be reused. The only distance to compute is $d(p, q)$.

If we suppose that all the elements were extracted i.i.d. from an unknown probability distribution, the the probability of being q the i-pivot is $\frac{1}{n}$.

If q is the new i pivot, the cost of rebuilding the table is at most $m + (m - i + 1)n$. Then, it is easy to find that the expected number of distance computations, due to updating the table, when inserting an object in an index with n objects can be bounded by:

$$E_\tau(n) \leq \sum_{i=2}^{m} \frac{1}{n} (m + (m - i + 1)n) \leq \frac{m(m+1)}{2}$$

Surprisingly, after some cancellations, the bound finally does not depends on the database size.

B. Updating the tree

The strategy to update the tree is similar to the one used updating the table.

The idea is to follow the same algorithm that was used to build the old tree but stopping if q, the object to be inserted, is selected as representative. In such case this branch of the tree is rebuilt.

Inserting q into a leaf node is trivial. inserting q into a node t with children l and r, has three possibilities (events) (see fig. **??**):

1) if $d(t_r, q) > t_\rho$, the subtree should be rebuilt,
2) if $d(l_r, q) < d(r_r, q)$, q is inserted in l,
3) if $d(l_r, q) \geq d(r_r, q)$, q is inserted in r.

In order to perform the complexity analysis we have to make the assumption that the tree is not pathologically

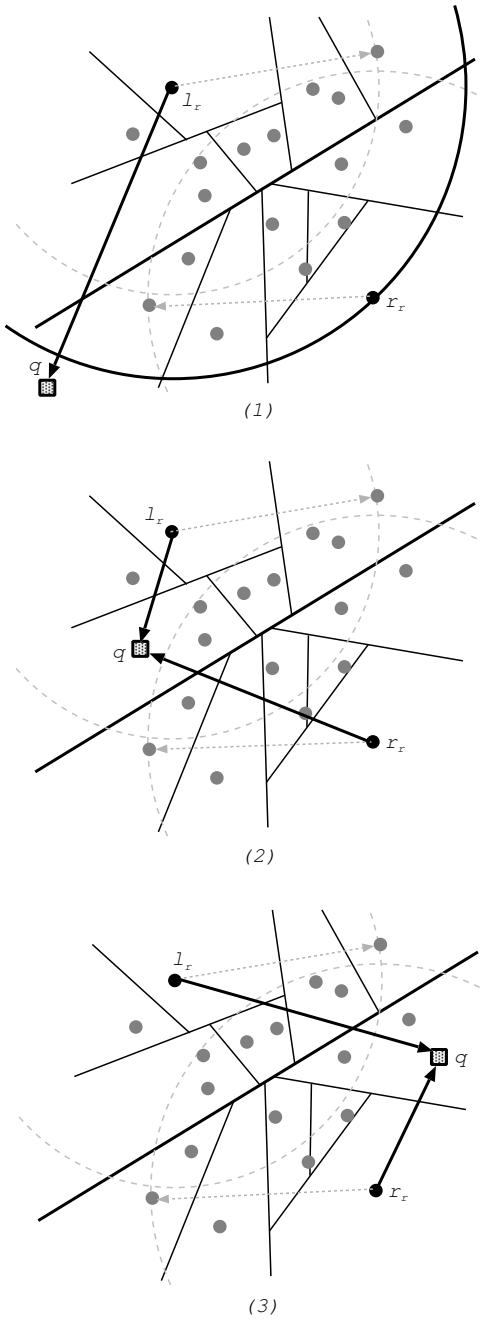

Fig. 1. Representation of the three possibilities when inserting an object in a node: the subtree should be rebuilt (1), the object is inserted in l node (2), and the object is inserted in r node (3).

unbalanced. That is, we suppose we can define an imbalance factor $\alpha \in [0.5, 1)$:

$$\alpha = \max_{t \in T} \frac{\max\{|l|, |r|\}}{|t|}$$

where T is the set of all the subtrees (with children), $|t|$ is the size (number of nodes) of the tree t, and l (r) is the left (right) subtree of t.

Then, the height h of the tree can be bounded by:

$$h < -\frac{\log(n)}{\log(\alpha)}$$

and the cost of building a MDF tree with n objects is bounded by $nh < -n\frac{\log(n)}{\log(\alpha)}$.

Now, as we are supposing the objects were extracted i.i.d from an unknown probability distribution, the probability of event **??**) is $\frac{1}{n}$, and the probabilities of **??**) or **??**) is $\frac{n-1}{n}$.

Therefore, the expected number of distance computations when inserting in a MDF tree with n objects can be expressed recursively as:

$$E_t(n) \leq 1 + \frac{1}{n} n \left(-\frac{\log(n)}{\log(\alpha)} \right) + \frac{n-1}{n} E_t(\alpha n)$$

Which leads to

$$E_t(n) \leq \frac{\log^2(n)}{2\log^2(\alpha)} - \frac{3\log(n)}{2\log(\alpha)}$$

C. Expected number of distance computations

Then, the expected number of distance computations when inserting in the TLAESA index is bounded by:

$$\begin{aligned} E(n) &= E_\tau(n) + E_t(n) \\ &\leq \frac{m(m+1)}{2} + \frac{\log^2(n)}{2\log^2(\alpha)} - \frac{3\log(n)}{2\log(\alpha)} \in O(\log^2(n)) \end{aligned}$$

Intuitively, what is happening here is that the high cost of rebuilding a large part of the structure is overcome by the low probability of the event.

In the next section we are going to design some experiments to illustrate the validity of this result.

IV. Experimental results

Due to its ubiquity, there exists fast search algorithm specialised in the Euclidean distance [**?**], [**?**]. TLAESA was not devised to compete with those algorithms, it was devised to work with very expensive distances (*i.e.* the edit distance), were the saving of a few distance computations can overcome the cost of maintaining complex data structures.

Nevertheless, we include some experiments with the Euclidean distance over a uniform distribution. This choice was made mainly because it is a well known distance and then, it is easy to understand what is happening. Furthermore, it is very easy to make increasingly difficult series of experiments.

We also make a series of experiments with real data and a non Euclidean distance (the edit distance) to show that the conclusions extracted from the Euclidean distance can be extrapolated.

All those experiments are addressed to show that our upper bound is accurate enough.

In this section, all the experiments follows the same pattern:

- First, a series of experiments with increasing number of pivots are performed in order to find the optimal number of pivots.

Fig. 2. Distance computations caused by an insertion for increasing size databases. Uniform distribution in the unit hypercube using the Euclidean distance in dimensions 5, 10, 15. Average of 10000 repetitions.

- Then, we perform one insertion in 10000 databases of each size. The sizes increase from 100 to a maximum that depends on the experiment in steps of 100. The average number of distance computations needed to insert an object for each size is plotted. To have an idea of the distribution of the number of distance computations, we also represent the 95% percentile, that is, in the 95% of the databases (of each size), the number of distances computations needed to update the index is bellow this value.
- As our bound depends of the imbalance factor α, a series of experiments was addressed to measure this value. For that, a series of MDF trees was generated from databases with sizes ranging equally than in the previous set of experiments. In this case, only 15 different databases were generated for each size. For each MDF tree we measured the imbalance factor and we used the average of all of them. Note that the imbalance factor depends on the maximum of a series of values (the tree node size ratio), then as the number of values increases (the size of the tree), it is more likely to find pathological high values of α. To avoid this effect, instead of finding the maximum, we find the 95% percentile. This value is called the reduced imbalance factor. Then the value of the reduced imbalance factor α_r, is lower than the imbalance factor (α). As a consequence, the value of the upper bound computed from α_r (showed in the plots) is lower than the upper bound computed from α. That means that the experimental average has not longer to be lower than the depicted upper bound, although in all the experiments it does.

A. Uniform distribution

As we said in the introduction of the section, this series of experiments is devoted to study the behaviour of our algorithm in an Euclidean space with artificial data drawn uniformly from hypercubes of dimensions 5, 10 and 15. First we performed some experiments to find, for each dimension, which is the optimum number of pivots. In this case we found that 14, 90 and 390 pivots are adequate for 5, 10 and 15 dimensions respectively (the experiments are not shown here because they are almost equal to the experiments that can be found in [**?**]).

Fig. **??** shows the results of the average number of distance computations needed to rebuild the index. The plots also shows the reduced imbalance factor in each case (0.81, 0.85 and 0.90 for dimension 5, 10 and 15 respectively) and the plotting of the upper bound.

Note that, in all the cases the average is below the upper bound. Moreover, the percentile 95% curve shows that, when the databases becomes larger, the event of having to rebuild big portions of the database are increasingly unlikely.

Fig. 3. Distance computations caused by an insertion for handwritten characters contour chains using the edit distance. Average of 10000 repetitions.

Fig. 4. Distance computations caused by an insertion for handwritten characters contour chains using the edit distance. Average of 10000 repetitions.

B. Contour chains

In order to assess the relevance of our model in a pattern recognition task, we applied it on the real world problem of handwritten digit classification. We used the NIST Special Database 3 of the National Institute of Standards and Technology. This database consists in 128×128 bitmap images of handwritten digits and letters. In this series of experiments, we only focus on digits written by 100 different writers. Each class of digit (from 0 to 9) has about 1,000 instances, then the whole database we used contains about 10,000 handwritten digits. In our experiments each digit was coded as a contour chain [**?**] and as dissimilarity measure the edit distance [**?**] was used.

In this case, 300 pivots were used and the reduced imbalance factor was found to be $\alpha_r = 0.90$.

Figure **??** shows the result for increasing size databases (from 100 to 8000 in steps of 100). Once more, the average number of distance computations respect our upper bound and as well as in the case of the Euclidean distance, the percentile $95\,\%$ curve shows that the rebuilding of big portions of the database is increasingly unlikely as the data base grows.

C. English dictionary

A last experiment using an English dictionary was performed. An English dictionary of 69069 words was extracted from the dictionary of the GNU spell checker (`ftp.gnu.org/gnu/aspell/dict/0index.html`). The edit distance was used as dissimilarity measure.

Fig. **??** shows the average number of distance computations for databases ranging from 100 to 4000 strings. In this case 340 pivots were used and the reduced imbalance factor was found to be 0.89. As in the previous experiments, similar conclusions can be drawn: the average number of distance computations is bounded by our upper bound, and the number of big reconstructions practically disappear when the database grows.

V. Conclusions

In this work we have proposed a new algorithm that overcome the problem of inserting new elements into an existing TLAESA index with a very low mean computation cost, $O(\log^2(n))$. We have proved this result both, theoretically and experimentally.

This problem is important because it allows the use of this type of indexes in incremental learning frameworks. This technique has shown to be very appropriate because the index on which the insertion is performed has a very low probability of provoking large changes on it.

The main drawback of this algorithm is that, in the present form, it is not suitable when strict real time restrictions should be followed. The worst case complexity for an insertion can be as worse as $O(n \log n)$. Nevertheless, the algorithm provides an easy way to estimate its computation time, and then, we have the opportunity of delaying the insertion and store the object in an auxiliary structure.

Note that one of the objectives in this work was to obtain exactly the same index that would have been obtained if we had construct the index from scratch. In this case we avoid performances degradations due to the insertion. Following the previous procedure this is not assured, and we are going to have a trade off between efficiency in search time and efficiency in insertion time. Nevertheless, we think this worth being explored.

ACKNOWLEDGEMENTS

This work has been supported in part by grants TIN2009-14205-C04-01 from the Spanish CICYT (Ministerio de Ciencia e Innovación), and the Consellería d' Educació de la Comunitat Valenciana through project PROMETEO/2012/017.