A Constant average time algorithm to allow insertions in the LAESA fast Nearest Neighbour Search index

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

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

Abstract

Nearest Neighbour search is a widely used technique in Pattern Recognition. In order to speed up the search many indexing techniques have been proposed. However, most of the proposed techniques are static, that is, once the index is built the incorporation of new data is not possible unless a costly rebuilt of the index is performed. The main effect is that changes in the environment are very costly to be taken into account.

In this work, we propose a technique to allow the insertion of elements in the LAESA index. The resulting index is exactly the same as the one that would be obtained by building it from scratch. In this paper we also obtain an upper bound for its expected running time. Surprisingly, this bound is independent of the database size.

1. Introduction

LAESA [4] is a fast Nearest Neighbour search algorithm. It is simple to implement and is able to find the nearest neighbour computing, in average, a constant number of distance computations. The algorithm uses a linear size structure (the LAESA index) that is built during the preprocessing phase. The LAESA has a very wide application range since the only property demanded to the dissimilarity function is to fulfil the triangular inequality ([1]). This property makes the algorithm to be relevant when large databases and expensive distance computations are involved.

Unfortunately the LAESA index is static, that is, the insertion or deletion of one element to the index requires a complete rebuilding of the index. This is a serious drawback when these techniques should be applied to interactive systems [5].

In this work we propose an algorithm that allows the insertion of elements in constant expected time to the LAESA index. The index obtained after the insertion is the same as the one obtained if a complete rebuild would be made. This bound is theoretically derived and some experiments are performed to illustrate its validity.

2. The LAESA index

The LAESA [4] computes an index during the preprocess to speed up the search. This index is made of two components:

- a subset B (pivots) of the elements in a database D,
- a table T that stores the distances ($d(\cdot,\cdot)$) from any pivot to any element in the database.

Micó *et al.* [4] showed that the optimal pivot set size ($|B|$) does not depend on the database size, but it depends only on the dissimilarity function. This is the unique parameter to be fixed in order to use the LAESA.

In order to simplify the notation, given a set of elements S and an element p, let we denote $d(S,p) = \min_{s\in S} d(s,p)$.

In the static algorithm the pivots are selected incrementally among the elements in the database. Although several methods are proposed for selecting the pivots [2], we are going to use the so called *maxmin* (algorithm 1) that has been used with excellent results:

- the first pivot (b_1) is randomly chosen. Let $B_1 = \{b_1\}$,
- the following pivot (b_i) is chosen incrementally as the farthest element to the actual pivot set ($b_i = \operatorname{argmax}_{p\in D} d(B_{i-1},p)$). Let $B_i = B_{i-1} \cup \{b_i\}$.

Given a database D and the size k for the pivot set, the pivot set is defined as $B = B_k = \{b_1,\ldots,b_k\}$. The

Algorithm 1: **build_LAESA_index(**D**,** k**)**

Input:
$D = \{p_1, \ldots, p_n\}$: database
k: number of pivots
Output:
$B = \{b_1, \ldots, b_k\}$: k size pivot set
$T : B \times D$ distance table
begin
b = random(D)
$B = \{b\}$
foreach $p \in D$ **do** $T[b, p] = d(b, p)$
for $i = 2$ **to** k **do**
$b = argmax_{p \in (D-B)} d(B, p)$
foreach $p \in D$ **do** $T[b, p] = d(b, p)$
$B = B \cup \{b\}$
end
end

table T will store the distance from each pivot to each element in D. It is obvious that this table can be built performing $|B||D|$ distance computations at most.

2.1. The dynamic case

Suppose we have an index (B, T) and we want to add a new element p to the index. The aim of the dynamic algorithm is to check iteratively (from $i = 2$ to k) which should be the i-th pivot, b_i or p.

- If it is b_i then almost no modifications to the table are needed (just add $d(b_{i-1}, p)$ to the table) and we test for the next i.
- If it is p, the rest of the table and the pivot set are recomputed.

The proposed method can be found in algorithm 2.

Suppose x is going to be selected as the i-th pivot, then:

- $i - 1$ distances should be computed to find that x is really the i-th pivot (**while** loop),
- at most $(|B| - i + 1)(|D| + 1)$ distances are computed when rebuilding the rest of the table (**foreach** loops).

That is, a total of $(|B| - i + 1)|D| + |B|$ distances are computed.

The worst case is when x is the second pivot. In this case $(|B|-1)|D|+|B|$ distances are computed (it grows linearly with the database size). The best case, is when x is not selected as pivot. In this case only $|B|$ distances are computed (then, it does not depend of the database size).

Algorithm 2: **update_LAESA_index(**x**,** D**,** B**,** T**)**

Input:
x: element to insert
$D = \{p_1, \ldots, p_n\}$: database
$B = \{b_1, \ldots, b_k\}$: pivot set
T: $B \times D$ distance table
Output:
D': database
B': k size pivot set
T': $B' \times D'$ table of distances
begin
$T' = T$ `// extend T one column`
$D' = D \cup \{x\}$
$B' = \{b_1\}$
$i = 2$
while $d(B', x) \leq d(B', b_i)$ ***and*** $i \leq k$ **do**
$B' = B' \cup \{b_i\}$
$T'[b_i, x] = d(b_i, x)$
$i = i + 1$
end
if $i \leq k$ **then**
$B' = B' \cup \{x\}$
foreach $p \in D'$ **do** $T[x, p] = d(x, p)$
for $j = i + 1$ **to** k **do**
$b = argmax_{p \in (D'-B')} d(B', p)$
foreach $p \in D$ **do** $T'[b, p] = d(b, p)$
$B' = B' \cup \{b\}$
end
end
end

In the next section we are going to see that, in the average case like in the best case, the expected time does no depend on the database size.

2.2. Average time complexity

In order to obtain the expected number of distance computations we have first to find the probability of being x the i-th pivot. Let us assume that all the elements in $D \cup \{p\}$ where extracted i.i.d. from an unknown probability distribution. Note that once fixed the first pivot (b_1), the property "being the i-th pivot" establishes an order on the remaining elements. Then, the probability of being x the second pivot is exactly the same as being the third and so on[1]. As we have $|D|$ possibilities (b_1 is

[1] Note that this is only true if there are no ties for the candidates of being the i-th pivot. That is, for each step the farthest element to the actual pivot set is unique. If it is not the case, the algorithm is conservative and chooses the previously known element as pivot. Then, as x is the last element, in case of ties x is not selected, decreasing the probability of being the i-th pivot for small i. If we are in a space where the probability of ties is high (typically discrete spaces) the

fixed), the probability of being x the i-th pivot is $\frac{1}{|D|}$.

Now, the expected value can be computed as the sum of the probability of being the i-th pivot times its cost (for $i = 2$ to $|B|$) plus the probability of not being one of the $|B|$ pivots times its cost.

Formally:

$$E = \sum_{i=2}^{|B|} \frac{1}{|D|} \left[(|B| - i + 1)|D| + |B|\right] + \frac{|D| - |B| + 1}{|D|}|B| \quad (1)$$

$$= \frac{|B|(|B| + 1)}{2} \quad (2)$$

Surprisingly, this expression does not depends on the database size.

Intuitively, what is happening here is that pivots are very low probability elements. In the algorithm, the high cost of replacing a pivot is overcome by the low probability of finding a new one.

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

3. Experimental results

3.1. Uniform distribution

In this experiment, artificial data was extracted from a uniform distribution over a 5, 10 and 15 dimension unit hypercube. As it is usual using LAESA, previous experiments where performed to fix the pivot set size: 9, 50 and 270 pivots were use respectively.

Each experiment counts the number of distance computations needed to insert a random element in an existing index. Database sizes range from 100 to 10000 in steps of 100. For each database size, 10000 random databases were generated. Fig. 2 shows the results of the experiments.

For each database size, the average number of distance computations provoked by an insertion was represented. In order to have an idea of the distribution of the number of distance computations, the 95% percentile is also represented. That means the 95% of the insertions worked out a number of distance computations below the line. Note that in all cases the 95% percentile is a constant that corresponds to the number of pivots used in the experiment. In other words, in more than the 95% of the cases the number of distance computations is the lowest possible. The theoretical prediction of the expected value of the distance computations in equation 2 is also plotted. As it can be seen, the experimental results fits very well with the theoretical prediction.

expectation we get is an upper bound of the real one.

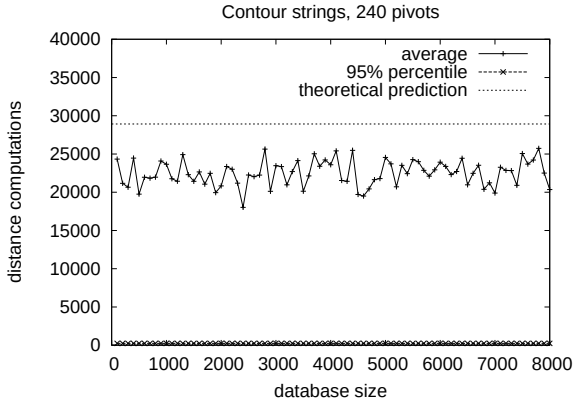

Figure 1. Distance computations caused by an insertion for handwritten characters contour chains using the edit distance. Average of 10000 repetitions.

3.2. 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 [6] and as dissimilarity measure the edit distance [3] was used. Note that as the edit distance can not be bigger than the longest contour chain, we are going to have many ties in the distances and then, as stated in the footnote 1, our theoretical prediction becomes an upper bound.

Figure 1 shows the result for increasing size databases (from 100 to 8000 in steps of 100). Each point is the average of 10000 random databases of the given size. Form 100 to 8000 in steps of 100 sizes for the databases where used.

4. Conclusions

In this work we have addressed the problem of inserting new elements into an existing LAESA index. We have proposed an algorithm that works in average

constant time, and we have proved this result both, theoretically and experimentally. This problem is so important because it opens the door to use this type of indexes in incremental learning frameworks.

The technique introduced here is clearly extensible to other ”robust” indexes. That is, when it is very unlikely that the addition of new elements provokes a big changes in the index.

Moreover, the algorithm can be largely improved by introducing some heuristics *i.e.* do not change the actual pivot if the new candidate is not further than a threshold. Although in such cases the resulting index is not exactly the same as the one that would be obtained building it from scratch and some trade off between insertion speed and search speed is introduced.

5. Acknowledgements

This work has been supported in part by grants TIN2009-14205-C04-01 from the Spanish CICYT (Ministerio de Ciencia e Innovación), 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. D. Blasi, and D. Reforgiato. Advanced indexing schema for imaging applications: three case studies. *Image Processing, IET*, 1(3):249–268, 2007.

[2] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recogn. Lett.*, 24(14):2357–2366, 2003.

[3] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and revers als. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[4] 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.

[5] G. P. Nguyen and M. Worring. Interactive access to large image collections using similarity-based visualization. *J. Vis. Lang. Comput.*, 19(2):203–224, 2008.

[6] T. T. O. Due Trier, Anil K. Jain. Feature extraction methods for character recognition - a survey. *Pattern Recognition*, 29(4):641–662, 1996.

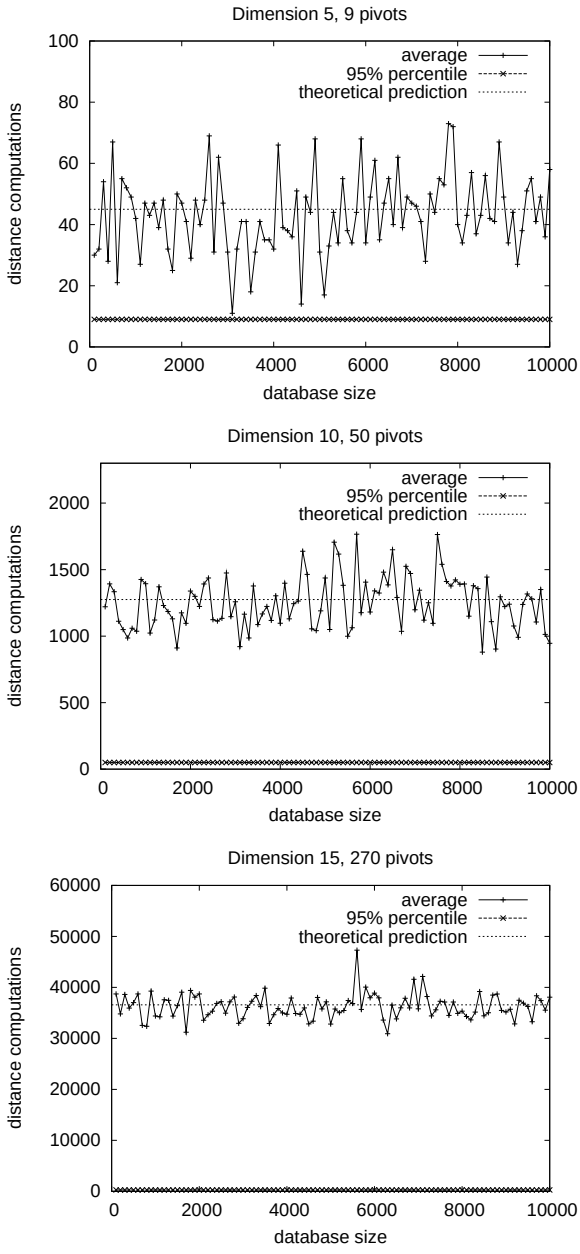

Figure 2. Distance computations caused by an insertion for increasing size databases. Uniform distribution in dimensions 5, 10, 15 unit hypercube. Average of 10000 repetitions.