Improvements of TLAESA Nearest Neighbour Search Algorithm and Extension to Approximation Search

Ken Tokoro **Kazuaki Yamaguchi** **Sumio Masuda**

Kobe University,
1-1, Rokkodai, Nada-ku, Kobe 657-8501 Japan,
Email: `ky@kobe-u.ac.jp`

Abstract

Nearest neighbour (NN) searches and k nearest neighbour (k-NN) searches are widely used in pattern recognition and image retrieval. An NN (k-NN) search finds the closest object (closest k objects) to a query object. Although the definition of the distance between objects depends on applications, its computation is generally complicated and time-consuming. It is therefore important to reduce the number of distance computations. TLAESA (Tree Linear Approximating and Eliminating Search Algorithm) is one of the fastest algorithms for NN searches. This method reduces distance computations by using a branch and bound algorithm. In this paper we improve both the data structure and the search algorithm of TLAESA. The proposed method greatly reduces the number of distance computations. Moreover, we extend the improved method to an approximation search algorithm which ensures the quality of solutions. Experimental results show that the proposed method is efficient and finds an approximate solution with a very low error rate.

Keywords: Nearest Neighbour Search, k Nearest Neighbour Search, TLAESA, Approximation Search, Distance Computaion.

1 Introduction

NN and k-NN searches are techniques which find the closest object (closest k objects) to a query object from a database. These are widely used in pattern recognition and image retrieval. We can see examples of their applications to handwritten character recognition in (Rico-Juan & Micó 2003) and (Micó & Oncina 1998), and so on. In this paper we consider NN (k-NN) algorithms that can work in any metric space. For any x, y, z in a metric space, the distance function $d(\cdot,\cdot)$ satisfies the following properties:

$$d(x,y) = 0 \Leftrightarrow x = y,$$
$$d(x,y) = d(y,x),$$
$$d(x,z) \leq d(x,y) + d(y,z).$$

Although the definition of the distance depends on applications, its calculation is generally complicated and time-consuming. We particularly call the calculation of $d(\cdot,\cdot)$ a *distance computation.*

For the NN and k-NN searches in metric spaces, some methods that can manage a large set of objects efficiently have been introduced(Hjaltason & Samet 2003). They are categorized into two groups. The methods in the first group manage objects with a tree structure such as vp-tree(Yianilos 1993), M-tree(Ciaccia, Patella & Zezula 1997), sa-tree (Navarro 2002) and so forth. The methods in the second group manage objects with a distance matrix, which stores the distances between objects. The difference between two groups is caused by their approaches to fast searching. The former aims at reducing the computational tasks in the search process by managing objects effectively. The latter works toward reducing the number of distance computations because generally their costs are higher than the costs of other calculations. In this paper we consider the latter approach.

AESA (Approximating and Eliminating Search Algorithm)(Vidal 1986) is one of the fastest algorithms for NN searches in the distance matrix group. The number of distance computations is bounded by a constant, but the space complexity is quadratic. LAESA (Linear AESA)(Micó, Oncina & Vidal 1994) was introduced in order to reduce this large space complexity. Its space complexity is linear and its search performance is almost the same as that of AESA. Although LAESA is more practical than AESA, it is impractical for a large database because calculations other than distance computations increase. TLAESA (Tree LAESA)(Micó, Oncina & Carrasco 1996) is an improvement of LAESA and reduces the time complexity to sublinear. It uses two kinds of data structures: a distance matrix and a binary tree, called a search tree.

In this paper, we propose some improvements of the search algorithm and the data structures of TLAESA in order to reduce the number of distance computations. The search algorithm follows the best first algorithm. The search tree is transformed to a multiway tree from a binary tree. We also improve the selection method of the root object in the search tree. These improvements are simple but very effective. We then introduce the way to perform a k-NN search in the improved TLAESA. Moreover, we propose an extension to an approximation search algorithm that can ensure the quality of solutions.

This paper is organized as follows. In section 2, we describe the details of the search algorithm and the data structures of TLAESA. In section 3, we propose some improvements of TLAESA. In section 4, we present an extension to an approximation search algorithm. In section 5, we show some experimental results. Finally, in section 6, we conclude this paper.

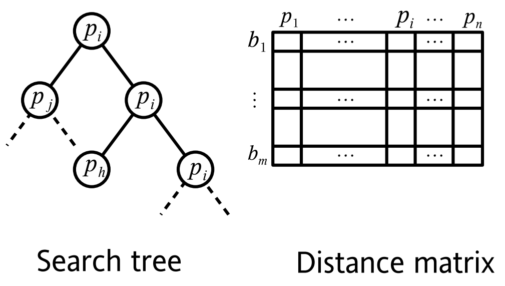

Figure 1: An example of the data structures in TLAESA.

2 TLAESA

TLAESA uses two kinds of data structures: the distance matrix and the search tree. The distance matrix stores the distances from each object to some selected objects. The search tree manages hierarchically all objects. During the execution of the search algorithm, the search tree is traversed and the distance matrix is used to avoid exploring some branches.

2.1 Data Structures

We explain the data structures in TLAESA. Let P be the set of all objects and B be a subset consisting of selected objects called *base prototypes*. The distance matrix M is a two-dimensional array that stores the distances between all objects and base prototypes. The search tree T is a binary tree such that each node t corresponds to a subset $S_t \subset P$. Each node t has a pointer to the representative object $p_t \in S_t$ which is called a *pivot*, a pointer to a left child node l, a pointer to a right child node r and a covering radius r_t. The covering radius is defined as

$$r_t = \max_{p \in S_t} d(p, p_t). \tag{1}$$

The pivot p_r of r is defined as $p_r = p_t$. On the other hand, the pivot p_l of l is determined so that

$$p_l = \operatorname*{argmax}_{p \in S_t} d(p, p_t). \tag{2}$$

Hence, we have the following equality:

$$r_t = d(p_t, p_l). \tag{3}$$

S_t is partitioned into two disjoint subsets S_r and S_l as follows:

$$\begin{aligned} S_r &= \{p \in S_t | d(p, p_r) < d(p, p_l)\}, \\ S_l &= S_t - S_r. \end{aligned} \tag{4}$$

Note that if t is a leaf node, $S_t = \{p_t\}$ and $r_t = 0$. Fig. 1 shows an example of the data structures.

2.2 Construction of the Data Structures

We first explain the construction process of the search tree T. The pivot p_t of the root node t is randomly selected and S_t is set to P. The pivot p_l of the left child node and the covering radius r_t are defined by Eqs. (2) and (3). The pivot p_r of the right child node is set to p_t. S_t is partitioned into S_r and S_l by Eq. (4). These operations are recursively repeated until $|S_t| = 1$.

The distance matrix M is constructed by selecting base prototypes. This selection is important because base prototypes are representative objects which are used to avoid some explorations of the tree.

The ideal selection of them is that each object is as far away as possible from other objects. In (Micó et al. 1994), a greedy algorithm is proposed for this selection. This algorithm chooses an object that maximizes the sum of distances from the other base prototypes which have already been selected. In (Micó & Oncina 1998), another algorithm is proposed, which chooses an object that maximizes the minimum distance to the preselected base prototypes. (Micó & Oncina 1998) shows that the latter algorithm is more effective than the former one. Thus, we use the later algorithm for the selection of base prototypes.

The search efficiency depends not only on the selection of base prototypes but also on the number of them. There is a trade-off between the search efficiency and the size of distance matrix, i.e. the memory capacity. The experimental results in (Micó et al. 1994) show that the optimal number of base prototypes depends on the dimensionality dm of the space. For example, the optimal numbers are 3, 16 and 24 if $dm = 2, 4$ and 8, respectively. The experimental results also show that the optimal number does not depend on the number of objects.

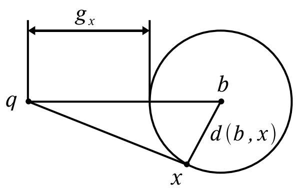

Figure 2: Lower bound.

2.3 Search Algorithm

The search algorithm follows the branch and bound strategy. It traverses the search tree T in the depth first order. The distance matrix M is referred whenever each node is visited in order to avoid unnecessary traverse of the tree T. The distance are computed only when a leaf node is reached.

Given a query object q, the distance between q and the base prototypes are computed. These results are stored in an array D. The object which is the closest to q in B is selected as the nearest neighbour candidate p_{min}, and the distance $d(q, p_{min})$ is recorded as d_{min}. Then, the traversal of the search tree T starts at the root node. The lower bound for the left child node l is calculated whenever each node t is reached if it is not a leaf node. The lower bound of the distance between q and an object x is defined as

$$g_x = \max_{b \in B} |d(q, b) - d(b, x)|. \tag{5}$$

See Fig. 2. Recall that $d(q, b)$ was precomputed before the traversals and was stored in D. In addition, the value $d(b, x)$ was also computed during the construction process and stored in the distance matrix M. Therefore, g_x is calculated without any actual distance computations. The lower bound g_x is not actual distance $d(q, x)$. Thus, it does not ensure that the number of visited nodes in the search becomes minimum. Though, this evaluation hardly costs, hence it is possible to search fast. The search process accesses the left child node l if $g_{p_l} \le g_{p_r}$, or the right child node r if $g_{p_l} > g_{p_r}$. When a leaf node is reached, the distance is computed and both p_{min} and d_{min} are updated if the distance is less than d_{min}.

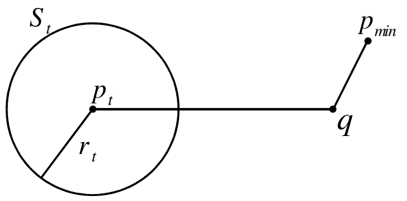

Figure 3: Pruning Process.

procedure NN search(q)

```
1:  $t \leftarrow$ root of $T$
2:  $d_{min} = \infty, g_{p_t} = 0$
3:  for $b \in B$ do
4:      $D[b] = d(q, b)$
5:      if $D[b] < d_{min}$ then
6:          $p_{min} = b, d_{min} = D[b]$
7:      end if
8:  end for
9:  $g_{p_t} = \max_{b \in B} |(D[b] - M[b, p_t])|$
10: search($t, g_{p_t}, q, p_{min}, d_{min}$)
11: return $p_{min}$
```

Figure 4: Algorithm for an NN search in TLAESA.

We explain the pruning process. Fig. 3 shows the pruning situation. Let t be the current node. If the inequality

$$d_{min} + r_t < d(q, p_t) \tag{6}$$

is satisfied, we can see that no object exists in S_t which is closer to q than p_{min} and the traversal to node t is not necessary. Since $g_{p_t} \leq d(q, p_t)$, Eq. (6) can be replaced with

$$d_{min} + r_t < g_{p_t}. \tag{7}$$

Figs. 4 and 5 show the details of the search algorithm(Micó et al. 1996).

3 Improvements of TLAESA

In this section, we propose some improvements of TLAESA in order to reduce the number of distance computations.

3.1 Tree Structure and Search Algorithm

If we can evaluate the lower bounds g in the ascending order of their values, the search algorithm runs very fast. However, this is not guaranteed in TLAESA since the evaluation order is decided according to the tree structure. We show such an example in Fig. 6. In this figure, u, v and w are nodes. If $g_{p_v} < g_{p_w}$, it is desirable that v is evaluated before w. But, if $g_{p_v} > g_{p_u}$, w might be evaluated before v.

We propose the use of a multiway tree and the best first order search instead of a binary tree and the depth first search. During the best first search process, we can traverse preferentially a node whose subset may contain the closest object. Moreover, we can evaluate more nodes at one time by using of the multiway tree. The search tree in TLAESA has many nodes which have a pointer to the same object. In the proposed structure, we treat such nodes as one node. Each node t corresponds to a subset $S_t \subset P$ and has a pivot p_t, a covering radius $r_t = \max_{p \in S_t} d(p, p_t)$ and pointers to its children nodes.

procedure search($t, g_{p_t}, q, p_{min}, d_{min}$)

```
1:  if $t$ is a leaf then
2:      if $g_{p_t} < d_{min}$ then
3:          $d = d(q, p_t)$ {distance computation}
4:          if $d < d_{min}$ then
5:              $p_{min} = p_t, d_{min} = d$
6:          end if
7:      end if
8:  else
9:      $r$ is a right child of $t$
10:     $l$ is a left child of $t$
11:     $g_{p_r} = g_{p_t}$
12:     $g_{p_l} = \max_{b \in B} |(D[b] - M[b, p_t])|$
13:     if $g_{p_l} < g_{p_r}$ then
14:         if $d_{min} + r_l > g_{p_l}$ then
15:             search($l, g_{p_l}, p_{min}, d_{min}$)
16:         end if
17:         if $d_{min} + r_r > g_{p_r}$ then
18:             search($r, g_{p_r}, p_{min}, d_{min}$)
19:         end if
20:     else
21:         if $d_{min} + r_r > g_{p_r}$ then
22:             search($r, g_{p_r}, p_{min}, d_{min}$)
23:         end if
24:         if $d_{min} + r_l > g_{p_l}$ then
25:             search($l, g_{p_l}, p_{min}, d_{min}$)
26:         end if
27:     end if
28: end if
```

Figure 5: A recursive procedure for an NN search in TLAESA.

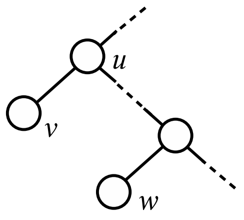

Figure 6: A case in which the search algorithm in TLAESA does not work well.

We show a method to construct the tree structure in Fig. 7. We first select randomly the pivot p_t of the root node t and set S_t to P. Then we execute the procedure **makeTree**(t, p_t, S_t) in Fig. 7.

We explain the search process in the proposed structure. The proposed method maintains a priority queue Q that stores triples (node t, lower bound g_{p_t}, covering radius r_t) in the increasing order of $g_{p_t} - r_t$. Given a query object q, we calculate the distances between q and base prototypes and store their values in D. Then the search process starts at the root of T. The following steps are recursively repeated until Q becomes empty. When t is a leaf node, the distance $d(q, p_t)$ is computed if $g_{p_t} < d_{min}$. If t is not a leaf node and its each child node t' satisfies the inequality

$$g_{p_t} < r_{t'} + d_{min}, \tag{8}$$

the lower bound $g_{p_{t'}}$ is calculated and a triple $(t', g_{p_{t'}}, r_{t'})$ is added to Q. Figs. 8 and 9 show the details of the algorithm.

procedure makeTree(t, p_t, S_t)

```
$t' \leftarrow$ new child node of $t$
if $|S_t| = 1$ then
    $p_{t'} = p_t$ and $S_{t'} = \{p_{t'}\}$
else
    $p_{t'} = \operatorname*{argmax}_{p \in S_t} d(p, p_t)$
    $S_{t'} = \{p \in S_t | d(p, p_{t'}) < d(p, p_t)\}$
    $S_t = S_t - S_{t'}$
    makeTree$(t', p_{t'}, S_{t'})$
    makeTree$(t, p_t, S_t)$
end if
```

Figure 7: Method to construct the proposed tree structure.

procedure NN search(q)

```
$t \leftarrow$ root of $T$
$d_{min} = \infty, g_{p_t} = 0$
for $b \in B$ do
    $D[b] = d(q, b)$
    if $D[b] < d_{min}$ then
        $p_{min} = b, d_{min} = D[b]$
    end if
end for
$g_t = \max_{b \in B} |(D[b] - M[b, p_t])|$
$Q \leftarrow \{(t, g_{p_t}, r_t)\}$
while $Q$ is not empty do do
    $(t, g_{p_t}, r_t) \leftarrow$ element in $Q$
    search$(t, g_{p_t}, q, p_{min}, d_{min})$
end while
return $p_{min}$
```

Figure 8: Proposed algorithm for an NN search.

3.2 Selection of Root Object

We focus on base prototypes in order to reduce node accesses. The lower bound of the distance between a query q and a base prototype b is

$$\begin{aligned} g_b &= \max_{b \in B} |d(q, b) - d(b, b)| \\ &= d(q, b). \end{aligned}$$

This value is not an estimated distance but an actual distance.

If we can use an actual distance in the search process, we can evaluate more effectively which nodes are close to q. This fact means that the search is efficiently performed if many base prototypes are visited in the early stage. In other words, it is desirable that more base prototypes are arranged in the upper part of the search tree. Thus, in the proposed algorithm, we choose the first base prototype b_1 as the root object.

3.3 Extension to a k-NN Search

LAESA was developed to perform NN searches and (Moreno-Seco, Micó & Oncina 2002) extended it so that k-NN searches can be executed. In this section, we extend the improved TLAESA to a k-NN search algorithm. The extension is simple modifications of the algorithm described above. We use a priority queue V for storing k nearest neighbour candidates and modify the definition of d_{min}. V stores pairs (object p, distance $d(q, p)$) in the increasing order of

procedure search$(t, g_{p_t}, q, p_{min}, d_{min})$

```
if $t$ is a leaf then
    if $g_{p_t} < d_{min}$ then
        $d = d(q, p_t)$ {distance computation}
        if $d < d_{min}$ then
            $p_{min} = p_t, d_{min} = d$
        end if
    end if
else
    for each child $t'$ of $t$ do
        if $g_{p_t} < r_{t'} + d_{min}$ then
            $g_{p_{t'}} = \max_{b \in B} |(D[b] - M[b, p_{t'}])|$
            $Q \leftarrow Q \cup \{(t', g_{p_{t'}}, r_{t'})\}$
        end if
    end for
end if
```

Figure 9: A procedure used in the proposed algorithm for an NN search.

procedure k-NN search(q, k)

```
$t \leftarrow$ root of $T$
$d_{min} = \infty, g_{p_t} = 0$
for $b \in B$ do
    $D[b] = d(q, b)$
    if $D[b] < d_{min}$ then
        $V \leftarrow V \cup \{(b, D[b])\}$
        if $|V| = k + 1$ then
            remove $(k + 1)$th pair from $V$
        end if
        if $|V| = k$ then
            $(c, d(q, c)) \leftarrow k$th pair of $V$
            $d_{min} = d(q, c)$
        end if
    end if
end for
$g_{p_t} = \max_{b \in B} |(D[b] - M[b, p_t])|$
$Q \leftarrow \{(t, g_{p_t}, r_t)\}$
while $Q$ is not empty do
    $(t, g_{p_t}, r_t) \leftarrow$ element in $Q$
    search$(t, g_{p_t}, q, V, d_{min}, k)$
end while
return $k$ objects $\leftarrow V$
```

Figure 10: Proposed algorithm for a k-NN search.

$d(q, p)$. d_{min} is defined as

$$d_{min} = \begin{cases} \infty & (|V| < k) \\ d(q, c) & (|V| = k) \end{cases} \tag{9}$$

where c is the object of the kth pair in V.

We show in Figs. 10 and 11 the details of the k-NN search algorithm. The search strategy essentially follows the algorithm in Figs. 8 and 9, but the k-NN search algorithm uses V instead of p_{min}.

(Moreno-Seco et al. 2002) shows that the optimal number of base prototypes depends on not only the dimensionality of the space but also the value of k and that the number of distance computations increases as k increases.

4 Extension to an Approximation Search

In this section, we propose an extension to an approximation k-NN search algorithm which ensures the

procedure search$(t, g_{p_t}, q, V, d_{min}, k)$

```
1:  if $t$ is a leaf then
2:     if $g_{p_t} < d_{min}$ then
3:        $d = d(q, p_t)$ {distance computation}
4:        if $d < d_{min}$ then
5:           $V \leftarrow V \cup \{(p_t, d(q, p_t))\}$
6:           if $|V| = k + 1$ then
7:              remove $(k+1)$th pair from $V$
8:           end if
9:           if $|V| = k$ then
10:             $(c, d(q,c)) \leftarrow k$th pair of $V$
11:             $d_{min} = d(q, c)$
12:          end if
13:       end if
14:    end if
15: else
16:    for each child $t'$ of $t$ do
17:       if $g_{p_t} < r_{t'} + d_{min}$ then
18:          $g_{p_{t'}} = \max_{b \in B} |(D[b] - M[b, p_{t'}])|$
19:          $Q \leftarrow Q \cup \{(t', g_{p_{t'}}, r_{t'})\}$
20:       end if
21:    end for
22: end if
```

Figure 11: A procedure used in the proposed algorithm for a k-NN search.

quality of solutions. Consider the procedure in Fig. 11. We replace the 4th line with

if $d < \alpha \cdot d_{min}$ **then**

and the 17th line with

if $g_t < r_{t'} + \alpha \cdot d_{min}$ **then**

where α is real number such that $0 < \alpha \leq 1$. The pruning process gets more efficient as these conditions become tighter.

The proposed method ensures the quality of solutions. We can show the approximation ratio to an optimal solution using α. Let a be the nearest neighbour object and a' be the nearest neighbour candidate object. If our method misses a and give a' as the answer, the equation

$$g(q, a) \geq \alpha \cdot d(q, a') \tag{10}$$

is satisfied. Then a will be eliminated from targeted objects. Since $g(q, a) \leq d(q, a)$, we can obtain the following equation:

$$d(q, a') \leq \frac{1}{\alpha} d(q, a). \tag{11}$$

Thus, the approximate solution are suppressed by $\frac{1}{\alpha}$ times of the optimal solution.

5 Experiments

In this section we show some experimental results and discuss them. We tested on an artificial set of random points in the 8-dimensional euclidean space. We also used the euclidean distance as the distance function. We evaluated the number of distance computations and the number of accesses to the distance matrix in 1-NN and 10-NN searches.

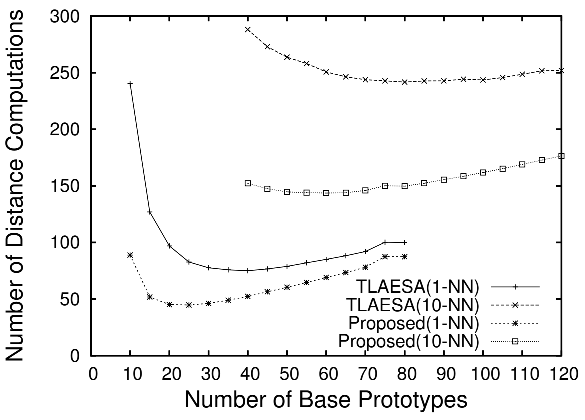

Figure 12: Relation of the number of distance computations to the number of base prototypes.

	1-NN	10-NN
TLAESA	40	80
Proposed	25	60

Table 1: The optimal number of base prototypes.

5.1 The Optimal Number of Base Prototypes

We first determined experimentally the optimal number of base prototypes. The number of objects was fixed to 10000. We executed 1-NN and 10-NN searches for various numbers of base prototypes, and counted the number of distance computations. Fig. 12 shows the results. From this figure, we chose the number of base prototypes as shown in Table. 1.

We can see that the values in the proposed method are fewer than those in TLAESA. This means that the proposed method can achieve better performance with smaller size of distance matrix. We used the values in Table. 1 in the following experiments.

5.2 Evaluation of Improvements

We tested the effects of our improvements described in 3.1 and 3.2. We counted the numbers of distance computations in 1-NN and 10-NN searches for various numbers of objects. The results are shown in Figs. 13 and 14. Similar to TLAESA, the number of the distance computations in the proposed method does not depend on the number of objects. In both of 1-NN and 10-NN searches, it is about 60% of the number of distance computations in TLAESA. Thus we can see that our improvements are very effective.

In the search algorithms of TLAESA and the proposed methods, various calculations are performed other than distance computations. The costs of the major part of such calculations are proportional to the number of accesses to the distance matrices. We therefore counted the numbers of accesses to the distance matrices. We examined the following two cases:

(i) TLAESA vs. TLAESA with the improvement of selection of the root object.

(ii) Proposed method only with improvement of tree structure and search algorithm vs. proposed method only with the improvement of selection of the root object.

In the case (i), the number of accesses to the distance matrix is reduced by 12% in 1-NN searches and 4.5% in 10-NN searches. In the case (ii), it is reduced by 6.8% in 1-NN searches and 2.7% in 10-NN searches.

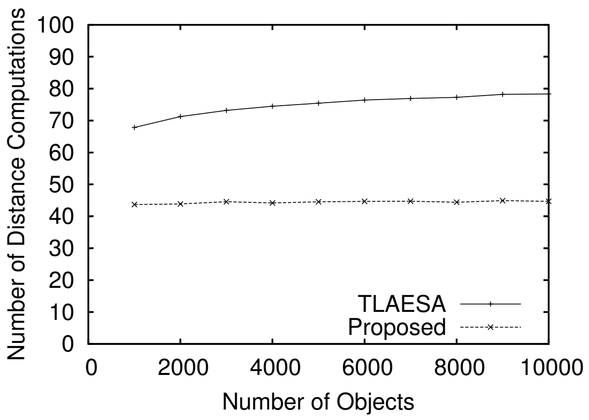

Figure 13: The number of distance computations in 1-NN searches.

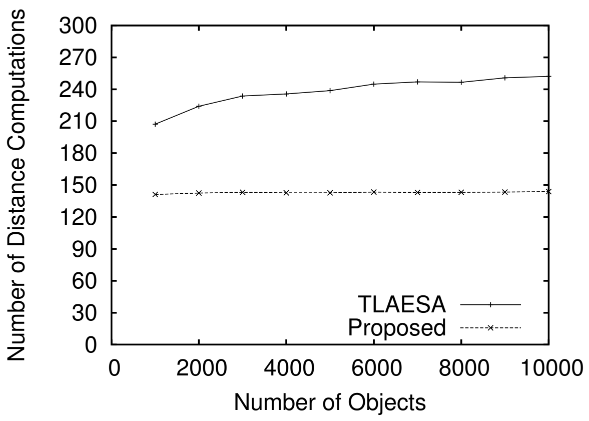

Figure 14: The number of distance computations in 10-NN searches.

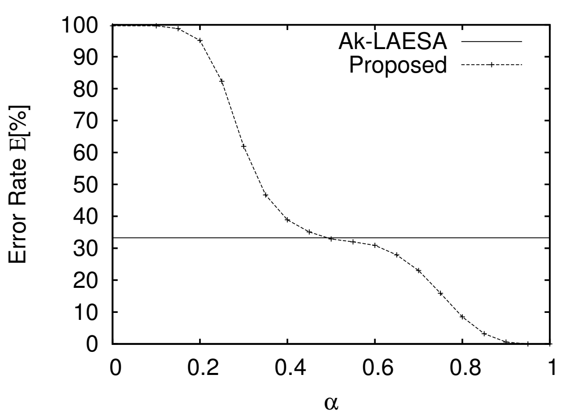

Figure 15: Error rate in 10-NN searches.

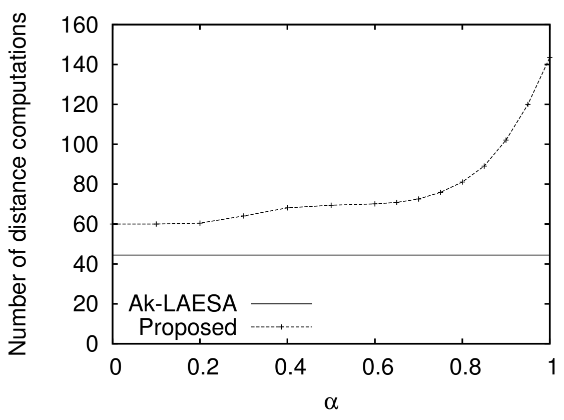

Figure 16: Relation of the number of distance computations to the value of α in 10-NN searches.

Thus we can see that this improvement about selection of the root object is effective.

5.3 Evaluation of Approximation Search

We tested the performance of the approximation search algorithm. We compared the proposed method to Ak-LAESA, which is the approximation search algorithm proposed in (Moreno-Seco, Micó & Oncina 2003). Each time a distance is computed in Ak-LAESA, the nearest neighbour candidate is updated and its value is stored. When the nearest neighbour object is found, the best k objects are chosen from the stored values. In Ak-LAESA, the number of distance computations of the k-NN search is exactly the same as that of the NN search.

To compare the proposed method with Ak-LAESA, we examined how many objects in the approximate solutions exist in the optimal solutions. Thus, we define the error rate E as follows:

$$E[\%] = \frac{|\{x_i | x_i \notin Opt, i = 1, 2, \cdots, k\}|}{k} \times 100 \quad (12)$$

where $\{x_1, x_2, \cdots, x_k\}$ is a set of k objects which are obtained by an approximation algorithm and Opt is a set of k closest objects to the query object.

Fig. 15 shows the error rate when the value of α is changed in 10-NN searches. Fig. 16 also shows the relation of the number of distance computations to the value of α in 10-NN searches. In the range $\alpha \geq 0.5$, the proposed method shows the lower error rate than Ak-LAESA. In particular, the error rate of the proposed method is almost 0 in range $\alpha \geq 0.9$. From two figures, we can control the error rate and the number of distance computations by changing the value of α. For example, the proposed method with $\alpha = 0.9$ reduces abount 28.6% of distance computations and its error rate is almost 0.

Then we examined the accuracy of the approximate solutions. We used $\alpha = 0.5$ for the proposed method because the error rate of the proposed method with $\alpha = 0.5$ is equal to the one of Ak-LAESA. We performed 10-NN searches 10000 times for each method and examined the distribution of kth approximate solution to kth optimal solution. We show the results in Figs. 17 and 18. In each figure, x axis represents the distance between a query object q and the kth object in the optimal solution. y axis shows the distance between q and the kth object in the approximate solution. The point near the line $y = x$ represents that kth approximate solution is very close to kth optimal solution. In Fig. 17, many points are widely distributed. In the worst case, some appriximate solutions reach about 3 times of the optimal solution. From these figures, we can see that the accuracy of solution by the proposed method is superior to the one by Ak-LAESA. We also show the result with $\alpha = 0.9$ in Fig. 19. Most points exist near the line $y = x$.

Though Ak-LAESA can reduce drastically the number of distance computations, its approximate solutions are often far from the optimal solutions. On the other hand, the proposed method can reduce the number of distance computations to some extent with

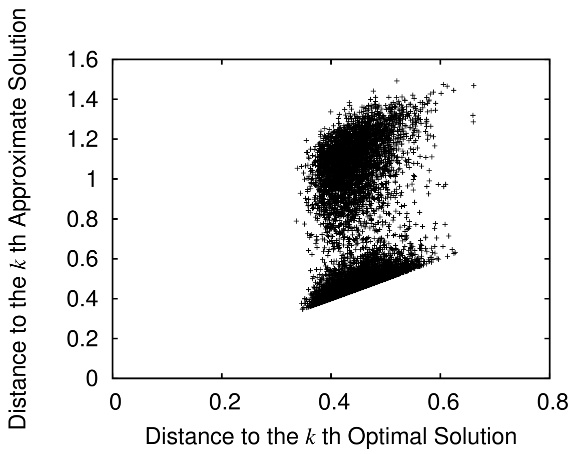

Figure 17: The distribution of the approximate solution by Ak-LAESA to the optimal solution.

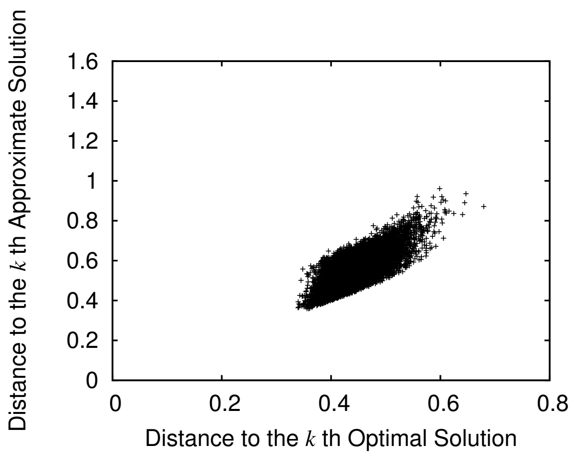

Figure 18: The distribution the approximate solution by the proposed method with $\alpha = 0.5$ to the optimal solution.

very low error rate. Moreover, the accuracy of its approximate solutions is superior to that of Ak-LAESA.

6 Conclusions

In this paper, we proposed some improvements of TLAESA. In order to reduce the number of distance computations in TLAESA, we improved the search algorithm to best first order from depth first order and the tree structure to a multiway tree from a binary tree. In the 1-NN searches and 10-NN searches in a 8-dimensional space, the proposed method reduced about 40% of distance computations. We then proposed the selection method of root object in the search tree. This improvement is very simple but is effective to reduce the number of accesses to the distance matrix. Finally, we extended our method to an approximation k-NN search algorithm that can ensure the quality of solutions. The approximate solutions of the proposed method are suppressed by $\frac{1}{\alpha}$ times of the optimal solutions. Experimental results show that the proposed method can reduce the number of distance computations with very low error rate by selecting the appropriate value of α, and that the accuracy of the solutions is superior to Ak-LAESA. From these viewpoints, the method presented in this paper is very effective when the distance computations are time-consuming.

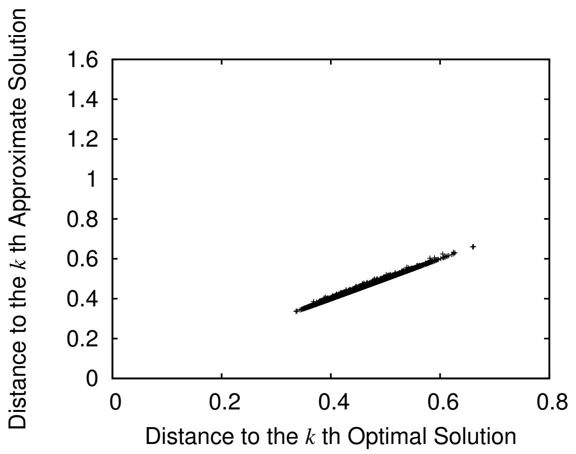

Figure 19: The distribution the approximate solution by the proposed method with $\alpha = 0.9$ to the optimal solution.

References

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

Hjaltason, G. R. & Samet, H. (2003), 'Index-driven similarity search in metric spaces', *ACM Transactions on Database Systems* **28**(4), 517–580.

Micó, L. & Oncina, J. (1998), 'Comparison of fast nearest neighbour classifiers for handwritten character recognition', *Pattern Recognition Letters* **19**(3-4), 351–356.

Micó, L., Oncina, J. & Carrasco, R. C. (1996), 'A fast branch & bound nearest neighbour classifier in metric spaces', *Pattern Recognition Letters* **17**(7), 731–739.

Micó, M. L., Oncina, J. & Vidal, E. (1994), 'A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements', *Pattern Recognition Letters* **15**(1), 9–17.

Moreno-Seco, F., Micó, L. & Oncina, J. (2002), 'Extending LAESA fast nearest neighbour algorithm to find the k-nearest neighbours', *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence* **2396**, 691–699.

Moreno-Seco, F., Micó, L. & Oncina, J. (2003), 'A modification of the LAESA algorithm for approximated k-NN classification', *Pattern Recognition Letters* **24**(1-3), 47–53.

Navarro, G. (2002), 'Searching in metric spaces by spatial approximation', *The VLDB Journal* **11**(1), 28–46.

Rico-Juan, J. R. & Micó, L. (2003), 'Comparison of AESA and LAESA search algorithms using string and tree-edit-distances', *Pattern Recognition Letters* **24**(9-10), 1417–1426.

Vidal, E. (1986), 'An algorithm for finding nearest neighbours in (approximately) constant average time', *Pattern Recognition Letters* **4**(3), 145–157.

Yianilos, P. N. (1993), Data structures and algorithms for nearest neighbor search in general metric spaces, *in* 'SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms', pp. 311–321.