PERGAMON

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Pattern Recognition 37 (2004) 1699 – 1711

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

A fast expected time algorithm for the 2-D point pattern matching problem

P.B. Van Wamelen[a,*], Z. Li[b], S.S. Iyengar[b,1]

[a]*Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803, USA*
[b]*Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA*

Received 27 May 2003; accepted 1 December 2003

Dedicated to Professor R.L. Kashyap on the occasion of his 61st birthday (Purdue University)

Abstract

Point set pattern matching is an integral part of many pattern recognition problems. We study a randomized algorithm for the alignment approach to model-based recognition.

Under certain mild assumptions we show that if our scene is a set of n points and our model is a set of $m < n$ points our algorithm has expected running time $O(n(\log m)^{3/2})$ for finding an occurrence of the model in the scene. This is significantly faster than any existing algorithms in the literature. We then describe some experimental results on randomly generated data using a practical version of our algorithm. These results agree well with the theoretical analysis.
© 2004 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.

Keywords: Point pattern matching expected time algorithm

1. Introduction

The design and analysis of data structures and algorithms is an important area of point pattern matching algorithms. In recent years there have been several significant advances in this area, ranging from $O(n^6)$ complexity to $O(n^2)$ algorithms. These advances have focused on developing faster algorithms and furthermore the results have kindled a lot of interest in obtaining an optimal algorithm in this area. These results have produced efficient solutions to many real-life problems. For details, see Refs. [1–8]. See also the survey article [9]. A comparison of these results is provided in Table 1.

Alignment [3] is one of the basic approaches to model-based object recognition. The method consists of choosing an ordered pair of points from the model and then, for every ordered pair of distinct points in the scene, computing a transformation mapping the model pair to the scene pair. This transformation is then tested to see whether it maps the entire model into the scene.

Apparently, the first researchers to consider randomization in connection with point pattern matching by alignment was Irani and Raghaven [5]. Their computational result is the most efficient in the literature. We will build on and improve their results. In all the situations they consider their algorithm always has running time $\Omega(n^2)$. This is because they always align two points in the model with two arbitrary points in the scene. As there are n^2 possible pairs of points in the scene and any of these can correspond to the model they all need to be tested. The main idea of this paper is to make use of the fact that we are looking for the model to occur in the scene without lots of extra scene points in between the images of the model points. That is, close neighbors in the model must map to close neighbors in the scene. We are looking for a telephone in a room, not a constellation in the

* Corresponding author. Tel.: +1-225-578-1675; fax: +1-225-578-4273.
E-mail addresses: wamelen@math.lsu.edu (P.B. Van Wamelen), zhili@bit.csc.lsu.edu (Z. Li), iyengar@bit.csc.lsu.edu (S.S. Iyengar).

[1] Partially supported by a DOE-ORNL grant.

Table 1
History of results on point set pattern matching algorithms

Year	Researcher	Technique	Geometric properties	Complexity[a]
1980	Ranade and Rosenfeld [1]	Relaxation approach	Translation differences	$O(n^4)$
1984	Ogawa [2]	Fuzzy relaxation	Translation, rotation, scaling differences	$O(n^6)$
1987	Huttenlocher and Ullman [3]	Alignment	Translation, rotation, scaling	$O(m^3n^2 \log n)$
1993	Vinod and Ghose [4]	Asymmetric neural networks	Translation, rotation, distortion and noise but not scaling	
1996	Irani and Raghavan [5]	Randomized alignment	Translation, rotation and scaling	$O(n^2 m \log n)$
1997	Chang et al. [6]	2-D Cluster approach	Translation, rotation, scale changes, local distortion, extra and or missing points	$O(n^4)$
1998	Boxer [7]	A sequential algorithm	Translations and rotation differences in 3D	$O(n^2(\lambda_6(n)/n)^{1/2}) \log n)$
1998	Chang et al. [8]	Nearest neighbors search	Translation, rotation, scaling, local distortion and extra/missing points	$O(k^2n^2)$
1999	van Wamelen et al	Probabilistic, sorted nearest neighbors	Translation, rotation, scaling, local distortion and extra/missing points	$O(n(\log m)^{3/2})$

[a]n is the number of points in the pattern to be matched.

night sky. This allows us to restrict the pairs of points in the scene that needs to be "aligned". The drawback of this method is that when we align points that are close neighbors the accuracy of the resulting transformation depends heavily on how accurately aligned the points were. That is, we have to be very careful of the errors in the location of matching points. This makes the algorithm and its analysis somewhat complicated.

2. Statement of the problem

Suppose two point patterns M and S in two dimensions are given. That is $M = \{p_1, p_2, \ldots, p_m\}$ and $S = \{q_1, q_2, \ldots, q_n\}$, where the p_i and q_j are points in $\mathbb{R}^2$. We will think of M as a model of some object and S as a scene in which the model may occur. In particular, n is bigger than or equal to m. We want to find a similarity transformation T_{s,θ,t_x,t_y}, such that $T(M)$ "matches" some subset of S, where matching will be made precise below. In the transformation T_{s,θ,t_x,t_y}, s is a scaling factor, θ a rotation angle and t_x and t_y the x and y translations, respectively. That is, for $(x, y) \in \mathbb{R}^2$, we have

$$T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

To define a match we assume two parameters, the matching probability, $\rho \in [0, 1]$ and the matching threshold, $t \in \mathbb{R}^+$ are given. Then we say $T_{s,\theta,t_x,t_y}(M)$ matches a subset of S (or occurs in S) if there exists a subset $M' \subset M$ with at least ρm elements such that for each $p \in M'$ we have $|T_{s,\theta,t_x,t_y}(p) - q| < t$ for some $q \in S$. We also assume that the image of the model does not contain many scene points that are not part of the model image. That is, we assume that the image (under the matching transformation) of the convex hull of the model contains at most m scene points (of which at least ρm match the model).

The use of a matching threshold will of course allow for a slightly "incorrect" match, but more importantly it allows for noise in the positions of the points, that is, a perturbation in the coordinates of the points to be matched.

Note that, as stated, it is trivial to solve the above problem: just transform all points in M to a disc of radius t around a single point in S. In order to avoid this we will require our match to not match any two model points to the same scene point.

What is a reasonable value for t? Suppose the radius of the set of points S is r (i.e., the point set S lies in a disk of radius r). Then, assuming uniform distribution of points, the average shortest distance to the nearest neighbor of a point in S is $r/(2\sqrt{n})$, see Section 4.1, Eq. (4). It seems reasonable to choose t to be some constant fraction of this distance. Let λ be this fraction, which will be called the matching factor. For the rest of the paper, we will assume that t was picked in this way, i.e.,

$$t = \lambda \frac{r}{2\sqrt{n}}. \tag{1}$$

In our run time analysis we will assume that λ stays constant as n grows. Note that in general when we think of larger point sets we think of them as also growing in size. That is, we think of r as growing like $\sqrt{n}$. In this case we are essentially keeping the error bound constant. On the other hand, if we keep r constant then the more points we have the closer together they lie (on average). In this case, if we did not let t become smaller, matching would eventually (for very large n) have no meaning because there would be many points within t of *any* position in S.

In our run time analysis, we will assume that the points in M and the non-matching points in S are uniformly distributed. Our algorithm will still work in most cases where this is not true, but it will fail (as would the alignment method in general) if the model has many regularly spaced points, that is when it is highly self-similar. See Ref. [5] and Section 5.3.

We are therefore assuming that S contains a locally distorted image of M under a similarity transformation with extra and/or missing points. Our problem is to recover the similarity transformation from the two sets of points. In this paper, we present a new technique for solving a problem of this nature.

3. The algorithm

The idea of the algorithm is that, with high probability, one of the first few random points in M will correspond to some point in S. So we take a random point in M and then search for a point in S such that it matches the point in M "locally". By that we mean that there exists a similarity transformation that maps the nearest neighbors of M to those of S. If we find such a map, it is easy to check whether it also gives a "global" match.

We now describe the algorithm more formally.

3.1. The main loop

The main loop is described in Fig. 1.

3.2. Finding points in S

The precomputation done in Step 1(b), see Fig. 1, allows one to check, in $O(1)$ time, whether there is a point in S within a short distance t_0 of any coordinate $q_0(x, y)$. By short we mean $t_0 < r/\sqrt{n}$. We simply find the square (j,k) of the look-up array in which the coordinate falls and then for each of the points, q', occurring in the 8 squares around (j,k) and the square (j,k) check for a match, that is, whether $|q_0 - q'| < t_0$. See Fig. 2.

3.3. Comparing nearest neighbors

This algorithm will depend on two parameters, k_2 and k_3. Guidelines for the asymptotically best values to choose can be found in the computational complexity section (Section 4), and some particular examples of good choices in the

```
input two point sets $M$ and $S$ and the parameters $k$, $\rho$ and $t$.
Step 1. Precomputations
    a) For each point in $M$ find the $k$ nearest neighbors (in order of close-
    ness) and store in nearest neighbor list. Do the same for $S$. This can
    be done using the algorithms described in [10] or [11].
    b) Divide the smallest square into which $S$ fits into a two dimensional
    array of squares of side length $r/\sqrt{n}$. Let each entry in the array contain
    a list of the points in $S$ that lie in that square. This will allow us to
    quickly check whether there is a point in $S$ at a given coordinate. See
    Section 3.2.
Step 2.
    until a global match is found
        select a point $p$ at random from $M$.
        for each $q_i \in S$
            Determine whether the $k$ nearest neighbors of $p$ matches
            the $k$ nearest neighbors of $q_i$ using the local matching al-
            gorithm to be described in Section 3.3. Let $T$ be the sim-
            ilarity transformation that gave the local match. Check
            whether $T$ can be improved to give a global match. See
            section 3.4.
Step 3.
    output $T$ (that is $s$, $\theta$, $t_x$ and $t_y$) and the matching pairs.
```

Fig. 1. The main loop.

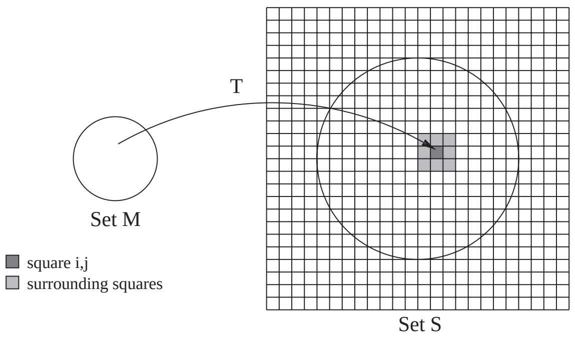

Fig. 2. Finding points in S.

implementation section (Section 5). The idea for checking whether the nearest neighbors of $p \in M$ match those of $q \in S$, is to assume that two close neighbors that are as far away as possible from p and q, respectively, correspond under a similarity transformation. We compute such a similarity transformation and then check whether a minimal number of the nearest neighbors match under this transformation. More precisely, assume that $\{a_1, a_2, \ldots, a_k\}$ are the k sorted nearest neighbors of the point p in M (with a_1 the nearest neighbor, etc.) and $\{b_1, b_2, \ldots, b_k\}$ the k sorted nearest neighbors of q in S. Then for each of the k_2 points $a_{k-[k_3/2]-k_2+1}$ to $a_{k-[k_3/2]}$ we compute the similarity transform T sending p to q and $a_{k-[k_3/2]-i}$, in turn, to each of the k_3 b's closest to $b_{k-[k_3/2]-i}$ (that is b_{k-k_3+1-i} to b_{k-i}). For each of these k_2k_3 T's we check whether they give more than $\rho(k-1)$ further nearest neighbor matches. For each T that does, we check whether that T can be refined to give a global match (see Fig. 3).

Note that the bigger we pick k, the further apart p and a_k are and the more accurate T is. On the other hand, if k is too big, the various errors may cause the ordering of corresponding nearest neighbors to be different in M and S. That is why we check k_2k_3 pairs of close neighbors.

Let $p = (p_x, p_y)$ and $a = (a_x, a_y)$ be two distinct points in M, and $q = (q_x, q_y)$ and $b = (b_x, b_y)$ be two distinct points in S. To find the unique similarity transformation, T, such that $T(p) = q$ and $T(a) = b$, we compute s, θ, t_x and t_y as follows:

$$s = \frac{|\overrightarrow{qb}|}{|\overrightarrow{pa}|},$$

$$\theta = \text{angle from } \overrightarrow{pa} \text{ to } \overrightarrow{qb},$$

$$t_x = q_x - p_x s\cos(\theta) + p_y s\sin(\theta),$$

$$t_y = q_y - p_x s\sin(\theta) - p_y s\cos(\theta). \tag{2}$$

As a practical improvement we might relax the condition "if there is a point, q', in S within t of $T(a_l)$" to the point q' being within a larger multiple of t of $T(a_l)$. In practice we used $|T(a_l) - q'| < 2t$. The reason for this is that T is computed from only two points and therefore even if all three image points q, b_{k-i-j} and q' are within t of their correct positions $T(a_l)$ might be further than t from q'. As will be seen in Eq. (6) of the run time analysis discussion (Section 4.2), the net effect is that constants in the O-notation will be altered, thereby producing only minor differences relative to complexity. It should be noted, however, that there is a substantial effect relative to the averaging algorithm for improving a local match to a global match, as described in Section 3.4.2 below, because there we assume that we start with a match that is within t. For purposes of the run time analysis we will therefore use t and not $2t$.

```
input Let $\{a_1, a_2, \ldots, a_k\}$ be the $k$ nearest neighbors of the point $p$ in
$M$. Let $\{b_1, b_2, \ldots, b_k\}$ be the $k$ nearest neighbors of $q$ in $S$.
for $i = k - k_2 + 1$ to $k$
    for $j = 0$ to $k_3 - 1$
        let $L_1 = \{p\}$ and $L_2 = \{q\}$.
        let $T$ be the unique transformation that sends $p$ to $q$ and $a_{i-[k_3/2]}$ to $b_{i-j}$.
        This $T$ is given by Equation (2).
        for $l = 1$ to $k$
            if there is a point, $q'$, in $S$ within $t$ of $T(a_l)$ (see section 3.2) then
                append $a_l$ to $L_1$ and append $q'$ to $L_2$.
        if (the number of points in $L_1$) $-2$ is greater than $\rho(k-1)$, then $T$ gives
        a local match.
            if, using the algorithm in Section 3.4 applied to $T$, this local match
            gives a global match
                return the transformation $T$ giving the global match and the matching
                points.
```

Fig. 3. Local matching algorithm.

```
input The two lists $L_1 = \{a_1, a_2, \ldots, a_l\}$ and $L_2 = \{b_1, b_2, \ldots, b_l\}$
until the number of matches is greater than $\rho n$.
    let $T$ be the transformation that gives the best least squares match between $L_1$ and $L_2$.
    let $L_1 = \{\}$ and $L_2 = \{\}$.
    unmark all $q$ in $S$.
    for $i = 1$ to $n$.
        if the algorithm in Section 3.2 finds a point within $t$ of $T(p_i)$ in $S$
            let $q \in S$ be the one closest to $T(p_i)$.
            if $q$ is not marked
                append $p_i$ to $L_1$ and append $q$ to $L_2$.
                mark $q$ in $S$.
    if $L_1$ is not longer than it was in the previous iteration
        abort, no global match found.
output $T$ and the matching pairs.
```

Fig. 4. Local to global by least squares.

3.4. *Finding a global match*

In this section we describe two algorithms, *least squares* and *averaging*, for improving a local match to a global match. The first algorithm is simpler, easier to implement and probably faster, but is hard to analyze. We therefore also give the second algorithm which is more involved but easier to analyze.

3.4.1. *Least squares*

Suppose T is given and it matches the points $\{a_1, a_2, \ldots, a_l\}$ in M and the points $\{b_1, b_2, \ldots, b_l\}$ in S (such that a_i matches b_i for each i). We start by refining the local match by computing the best least-squares match between the points that already match. That is, find the T that minimizes $\sum_{i=1}^{l} |T(a_i) - b_i|^2$. See, for example, Ref. [6] for explicit formulas. We then find all points matching under the refined T and repeat the process. We abort if we stop finding new matching points. We declare a global match if at some point we find more than ρn matches (see Fig. 4).

3.4.2. *Averaging*

The idea is to make use of the fact that under any similarity transform the average of n points gets mapped to the average of their images. This means that if we know that certain points correspond under a T_0 we can find a good approximation to it by selecting two regions B_1 and B_2 in M, computing the averages of points in B_1 and B_2 and the averages of their respective images in S and then finding the T sending the two averages to the image averages. Furthermore, if we know that a given T is fairly accurate we are guaranteed that if we match points with T, but with the matching distance relaxed, we will find *all* matching points. This leads to the algorithm below.

First note that the area common to two disks of radius t and with their centers a distance t apart is equal to 0.391 times the area of one of the disks. This implies that if, in a certain region, we check n_0 points to see if they match with points in S under T and $0.391\rho n_0$ of them do, then T must be within t of the correct similarity transform.

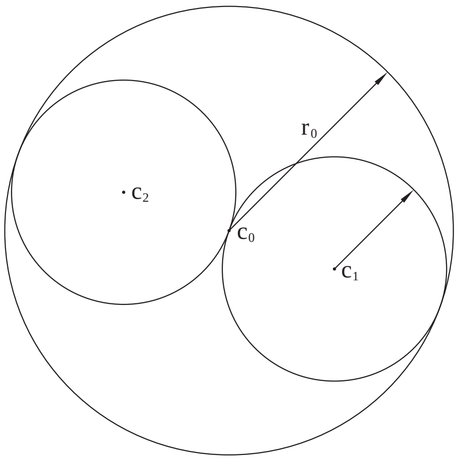

Fig. 5. The disks $B_{r_0}(c_0)$, $B_{r_0/2}(c_1)$ and $B_{r_0/2}(c_2)$.

Suppose T is given matching some points. Compute the center of mass of the matched points in S, say c_0. Now find r_0 such that everywhere in $B_{r_0}(c_0)$ (the ball of radius r_0 and center c_0), T is within t of the correct transformation. Such an r_0 can be found by checking the percentage of points that match in a region as explained in the previous paragraph.

Now pick a point c_1 in $B_{r_0}(c_0)$ such that $|c_1 - c_0| = r_0/2$ and let c_2 be the point on the opposite side of c_0 also at a distance $r_0/2$, see Fig. 5. For $i = 1$ or 2, let $L_i \subset M$ be all points that match within $2t$ with points in $S \cap B_{r_0/2}(c_i)$. Let $K_i \subset S$ be the points matching the points in L_i, that is $a_j \in L_i$

```
input A $T$ matching the two lists $L_0 = \{a_1, a_2, \ldots, a_l\}$ and $K_0 = \{b_1, b_2, \ldots, b_l\}$
    set $c_0 = \left(\sum_{j=1}^{l} b_j\right)/l$.
    find $r_0$ such that everywhere in $B_{r_0}(c_0)$ $40\rho\%$ of points match.
    if $r_0$ is not bigger than it was in the previous iteration
        abort no global match found.
    pick a point $c_1$ such that $|c_1 - c_0| = r_0/2$.
    let $c_2$ be the point on the opposite side of $c_0$ at the same distance. That is
        $c_2 = c_0 + (c_0 - c_1)$.
    let $L_1 = \{a_1, a_2, \ldots, a_{m_1}\} \subset M$ and $K_1 = \{b_1, b_2, \ldots, b_{m_1}\} \subset B_{r_0/2}(c_1)$
        be such that $|T(a_i) - b_i| < 2t$ for $i = 1, 2, \ldots, m_1$. We take these sets so large that
        no more matching points can be added and no two $a_i$ match the same $b$.
    let $L_2 \subset M$ and $K_2 \subset B_{r_0/2}(c_2)$ be defined similarly.
    let $l_i = \left(\sum_{a \in L_i} a\right)/m_i$ and $k_i = \left(\sum_{b \in K_i} b\right)/m_i$ for $i = 1, 2$ be the averages of the
        given sets.
    let $T$ be the unique linear transformation that sends $l_1$ to $k_1$ and $l_2$ to $k_2$.
repeat until a global match is found.
output $T$ and the matching pairs.
```

Fig. 6. Local to global by averaging.

and $b_j \in K_i$ satisfy $|T(a_j) - b_j| \leqslant 2t$. Let l_i be the average of the points in L_i and k_i be the average of the points in K_i. Finally, compute the similarity transform that sends l_1 to k_1 and l_2 to k_2. It will be shown below that this new T is a better match than the old T (in the sense that the new T will match points in a ball of radius αr_0 for some $\alpha > 1$). So repeating this procedure $O(\log m)$ times we will get a global match. See Fig. 6.

To find r_0 we could do the following. Divide the set M up into squares of side length some multiple of t. Then for each of these squares compute the percentage of points in it that match under T with points in S. Now find the biggest r_0 such that $B_{r_0}(c_0)$ contains only images of squares with more than $40\rho\%$ matched points. For this to work we will need the squares to be big enough that computing the percentage of matches is accurate but also small enough that there are enough squares to make r_0 accurate. We will therefore need m to be big. This is a big practical concern, but for purposes of asymptotic analysis no problem.

4. Computational complexity

4.1. Note on the expected distance to the kth nearest neighbor

In Ref. [10] it is proved that if we place n points randomly, with uniform distribution, in a disk of radius r then the expected distance from a given point to its kth nearest neighbor is given by

$$\frac{r}{\sqrt{\pi}} \frac{\Gamma(k+1/2)}{\Gamma(k)} \frac{1}{\sqrt{n}} \left(1 - \frac{3}{8n} + O\left(\frac{1}{n^2}\right)\right). \qquad (3)$$

In particular the expected distance to the nearest neighbor is

$$\frac{r}{2\sqrt{n}} + O\left(\frac{1}{n^{3/2}}\right). \qquad (4)$$

Stirling's formula implies that

$$\lim_{k\to\infty} \frac{\Gamma(k+1/2)}{\Gamma(k)\sqrt{k}} = 1. \qquad (5)$$

In particular

$$\frac{\Gamma(k+1/2)}{\Gamma(k)} = O(\sqrt{k})$$

and a good approximation for the distance to the kth nearest neighbor is therefore

$$\frac{r}{\sqrt{\pi}}\sqrt{\frac{k}{n}}.$$

4.2. Expected running time

We assume m is smaller than or equal to n. We will study the running time of the algorithm as n and m goes to infinity but ρ and λ stays constant. In particular, as noted in the introduction, we are assuming that r is growing like $\sqrt{n}$.

The precomputation for finding the nearest neighbors is $O(n \log n + nk \log k)$. See Refs. [11,16].

Constructing the look-up table is clearly $O(n)$.

Suppose that the probability of finding a local match between the k nearest neighbors of a random point $p_i \in M$ and the k nearest neighbors of one of the points in S, is ρ_0. Then the probability that we will check through l points in M in the main loop without getting a local match is $(1-\rho_0)^l$. So even for $\rho_0 = 0.6$, we should get a local match in 99% of cases after only 5 or less points in M ($0.4^5 = 0.01024$).

In fact we can compute the average number of points we will need to check in M until we find a local match in S. With probability ρ_0 we will need 1 point. With probability $(1-\rho_0)\rho_0$ we will need 2 points. With probability $(1-\rho_0)^{i-1}\rho_0$ we will need i points. So we will, on average, need

$$\rho_0 \sum_{i=1}^{\infty} (1-\rho_0)^{i-1} i = \rho_0 \frac{1}{\rho_0^2} = \frac{1}{\rho_0}$$

points.

So we see that, assuming there is a match, we will exit the *until* loop over M and S in Step 2 (Section 3.1) after, on average, n/ρ_0 executions of the body of the *until* loop.

What if there is no match? Assume that we have two point sets and we know that they either match with fixed parameters n, ρ and λ or that they do not match. Such a situation might for instance occur if we have a fingerprint and a large database of fingerprints to match against. In such a situation we can find a good value for ρ_0 by running the algorithm many times on data for which a match *does* exist and computing the average number of points in M that is checked before a correct local match is found. By the argument above we know that this value is $1/\rho_0$. Now, for any two point sets we can decide whether they match or not, as follows. Suppose we want the answer to be correct with probability x (say 0.99). Then find l such that $1-(1-\rho_0)^l > x$. Check the first l points in M for a local match. If this gives a global match then, of course, we know (with probability 1) that a match does exist. If, on the other hand, we do not find a match during the first l points, the probability of this happening (assuming there is a match) is very small, $(1-\rho_0)^l < 1-x$. So if we do not find a match within l points, we can say, with probability $1-(1-\rho_0)^l > x$ that in fact none exists.

Note that the probability ρ_0 is very close to, but less than, the ρ of the problem statement. This is because even if p does have a match in S, our algorithm for finding local matches is not guaranteed to find it. The probability of missing a correct local match will depend on our choice of k, k_2, and k_3. In some cases it might even speed up the overall algorithm by making choices for these parameters that make it more likely that we will miss a correct local match.

Note that checking whether a given similarity transformation T gives a global match takes time $\Omega(m)$. This means that we need to make sure that we do not find too many incorrect local matches. So let us see how big we need to choose k in order to find only $O(1)$ incorrect local matches while checking all the points in S against a particular point in M.

First note that the probability of finding a point within a distance t of a particular random position in S is

$$n\frac{\pi t^2}{\pi r^2} = n\frac{\pi\lambda^2 r^2}{4\pi n r^2} = \frac{\lambda^2}{4}. \qquad (6)$$

Suppose we pick a close neighbor of each of $p \in M$ and $q \in S$ and compute a similarity transformation, T, matching them. We then check whether there is a point in S within t of each of the k images $T(a_i)$, where the a_i are the k nearest neighbors of p. If we find ρk matches or more, we declare a success. What is the probability of an incorrect T passing this test? If we assume that each of the k tests are independent and using the probability found in the previous paragraph, we see that the probability of finding at least ρk matches in k tests is

$$\sum_{i=0}^{\rho k} \binom{k}{i} \left(\frac{\lambda^2}{4}\right)^i \left(1-\frac{\lambda^2}{4}\right)^{k-i}.$$

It is well known (see Refs. [12,13]) that this tail of the binomial distribution is bounded by

$$\mathrm{e}^{-2(\rho-\lambda^2/4)^2 k}.$$

Therefore, we will choose k to be bigger than

$$\frac{\ln m}{2(\rho-\lambda^2/4)^2}.$$

This ensures that we will get only $O(n/m)$ incorrect local matches in every n runs.

We need to choose k_2 and k_3 large enough such that we do not miss a local match if there is one. The problem is that because of the noise in the data the images of p's nearest neighbors in S might occur in a different order. Also the missing/extra points forces us to look at extra neighbors. By the same argument as for the number of points in M we will need to consider, we see that after checking k_2 of the nearest neighbors of p we will have found a point that does occur in S with probability $1-(1-\rho)^{k_2}$. We want this probability to be, say, 95%. This probability need not depend on the size of n because we only need this test to succeed once. We therefore choose k_2 approximately equal to $\log 0.05/\log(1-\rho)$, in particular k_2 is $O(1)$. On the other hand, the nearest neighbors changing their order is a more serious problem. By Eq. (3), the expected distance to the kth nearest neighbor of $q \in S$ is about

$$r_k = \frac{r}{\sqrt{\pi}} \frac{\Gamma(k+1/2)}{\Gamma(k)} \frac{1}{\sqrt{n}}.$$

A ring with outer radius $r_k + t$ and inner radius $r_k - t$ has area $4\pi r_k t$, and so we can expect there to be

$$n\frac{4\pi r_k t}{\pi r^2} = \frac{2\lambda}{\sqrt{\pi}} \frac{\Gamma(k+1)}{\Gamma(k)}$$

points in that ring. This means that the images of two consecutive nearest neighbors of p could be $O(\sqrt{k})$ apart in the sorted nearest neighbor list of q. We therefore choose k_3 approximately equal to $2\lambda\sqrt{k}/\sqrt{\pi}$ (see Eq. (5)).

Given T it takes time $O(k)$ to count the number of local matches. Thus we see that the local matching algorithm in Section 3.3 takes time $O(kk_2k_3) = O(k^{3/2})$.

By the argument in Section 4.3 below the total time for finding a global match from a local one is $O(m\log m)$.

Recall that we execute the inner loop in the algorithm of Section 3.3 a total of $nk_2k_3 = O(n(\log m)^{1/2})$ times, and that the probability that any one of them will succeed is $O(1/m)$ (by our choice of k). We therefore expect to have to check $O(n/m(\log m)^{1/2})$ local matches to see if they become global matches. Each such test takes time $O(m\log m)$. So this part of the algorithm takes time $O(n(\log m)^{3/2})$. The tests for local matches takes time $O(n(\log m)^{3/2})$ and the precomputations are $O(n\log n\log\log n)$. If m is larger than any root of n, that is $m > n^{\beta}$ for any $\beta < 1$, then $(\log m)^{3/2}$ will dominate $\log n\log\log n$ and so we see that our algorithm has expected running time

$$O(n(\log m)^{3/2}).$$

4.3. Local to global

In this subsection we want to analyze the averaging algorithm. In particular we want to show that in the algorithm of Section 3.4.2 we compute $O(\log m)$ T's before we have a global match. To do this we will show that, given a T with a certain r_0, the new T will have an expected r_0 equal to α times the old one for some constant $\alpha > 1$.

Recall our assumptions: we assume that there exists a T_0 (the transformation we are looking for) such that for ρm of the points a in M there is a point of S within t of $T_0(a)$ and that these points are uniformly distributed in the disk of radius t around $T_0(a)$. The other $(1-\rho)m$ points of M are missing in S and S also has some extra points. We will assume that in the region where M maps under T in S there are approximately $(1-\rho)m$ extra points and that these are uniformly distributed in this region.

First, let us compute the expected value of $|T(c_i)-T_0(c_i)|$ for the T given by applying one iteration of the *averaging* algorithm.

As explained in Section 3.4.2 the choice of r_0 means that for all $a\in M$ we have $|T(a)-T_0(a)| < t$, this means that if a corresponds to $b\in S$ then $|T(a)-b| \leqslant |T(a)-T_0(a)| + |T_0(a)-b| < 2t$ and so the sets K_1 and K_2 will contain all points that match under T_0. Unfortunately, they will also contain incorrect matches from the missing and extra points.

Let $d = n/(\pi r^2)$ be the density of points in S. Then the expected number of points in $B_{r_0/2}(c_i)$ is $N_0 = \pi(r_0/2)^2 d$.

For each of the image points under T there is a probability of

$$\frac{\pi(2t)^2}{\pi(r_0/2)^2}$$

that a given uniformly distributed point in $B_{r_0/2}(c_i)$ will be within $2t$ of the particular image. The probability that a given image point is within $2t$ of some point in $B_{r_0/2}(c_i)$ is therefore

$$1-\left(1-\frac{\pi(2t)^2}{\pi(r_0/2)^2}\right)^{N_0} < \pi(2t)^2 d = \lambda^2.$$

And so we can expect there to be a total of $N_2 = \lambda^2 N_0$ incorrect matches.

To count the number of correct matches we will assume that if e.g. $a\in M$ corresponds to $b\in S$ under T_0 but there is some incorrect $b'\in S$ such that $|T(a)-b'| < 2t$ then it is always closer to $T(a)$ than b is, so that we do not get the correct points matching up. The number of correct matches is then given by $N_1 = \rho(1-\lambda^2)N_0$.

If $L_i=\{a_1,a_2,\dots,a_{m_i}\}$, $K_i=\{b_1,b_2,\dots,b_{m_i}\}$ and l_i and k_i are the corresponding averages we want to find the expected value of

$$|k_i - T_0(l_i)| = \left|\left(\sum_{j=1}^{m_i}(b_j - T_0(a_j))\right)\Big/ m_i\right|.$$

For each of the N_1 j's corresponding to correct matches the $b_j - T_0(a_j)$ are uniformly distributed in a disk of radius t around 0. For the N_2 j's corresponding to incorrect matches the b_j's are uniformly distributed in a disk of radius $2t$ around $T(a_j)$. So, as a worst case, we will assume that every incorrect match introduces a systematic error of size t to the average. For instance, this will be the case if for all $a\in M$ $T(a)-T_0(a)$ is a constant of size t. The sum of all the $b_j - T_0(a_j)$ therefore has an expected value of $N_2 e$ for some $e\in\mathbb{R}^2$ with $|e| = t$. The average over all points then has an expected value of $N_2/m_i e$. To find the expected value of the absolute value of this quantity we need to consider also its variance. After all, if $|E(k_i - T_0(l_i))|$ is small but the standard deviation of $k_i - T_0(l_i)$ is large, $E(|k_i - T_0(l_i)|)$ will be much larger than $|E(k_i - T_0(l_i))|$. Fortunately, by the Central Limit Theorem, the average of m_i random variables each with variance σ^2 has variance σ^2/m_i. So by taking n large enough, k will be large and we will be computing an average over enough points to make the variance as small as we want compared to the expected value even in the first iteration of the algorithm. We can therefore take the expected value of $|k_i - T_0(l_i)|$ to be smaller than or equal to $N_2/m_i t$. Let $\gamma = N_2/m_i$ and note that $m_i = N_1 + N_2$ and so, by the formulas above,

$$\gamma = \frac{N_2}{m_i} = \frac{\lambda^2}{\rho(1-\lambda^2)+\lambda^2}.$$

So if ρ is large compared to λ^2 then the new transformation T (sending l_i to k_i) will be close to T_0 at c_1 and c_2. The lemma below quantifies how close this needs to be for the new T to be within t of T_0 in a disk around c_0 of radius strictly bigger than r_0. For a similar (but much harder) analysis of the errors introduced in computing an *affine* transform from points with errors see Refs. [14,15].

Lemma 1. *Suppose that T and T_0 are two linear transformations such that $|T(a_1) - T_0(a_1)| \leqslant \delta$ and $|T(a_2) - T_0(a_2)| \leqslant \delta$ for two points a_1 and a_2 and some $\delta\in\mathbb{R}^+$. Let $a_0 = (a_1 + a_2)/2$. Then for any a*

$$|T(a) - T_0(a)| < \sqrt{1+\left(\frac{|a-a_0|}{|a_1-a_0|}\right)^2}\,\delta.$$

Proof. If we think of the 2-D point sets as subsets of the complex numbers, then $T : \mathbb{C} \to \mathbb{C}$ can be written as

$$T(x) = t + sx$$

for some $t, s \in \mathbb{C}$. Let $v = a_1 - a_0$. For any $a \in \mathbb{C}$ write $a = a_0 + d_0 v$ for some $d_0 \in \mathbb{C}$. For any similarity transform T we then have

$$\begin{aligned} T(a) &= T(a_0 + d_0 v) \\ &= t + sa_0 + \frac{1}{2} d_0(2sv) \\ &= \frac{1}{2}(T(a_0 + v) + T(a_0 - v)) \\ &\quad + \frac{1}{2} d_0(T(a_0 + v) - T(a_0 - v)) \\ &= \frac{1 + d_0}{2} T(a_0 + v) + \frac{1 - d_0}{2} T(a_0 - v). \end{aligned}$$

So if we know that $|T(a_1) - T_0(a_1)| \leqslant \delta$ and $|T(a_2) - T_0(a_2)| \leqslant \delta$ we get, for $a = a_0 + d_0 v$,

$$\begin{aligned} |T(a) - T_0(a)| &= \left| \frac{1 + d_0}{2}(T(a_1) - T_0(a_1)) \right. \\ &\quad \left. + \frac{1 - d_0}{2}(T(a_2) - T_0(a_2)) \right| \\ &\leqslant \left(\left| \frac{1 + d_0}{2} \right| + \left| \frac{1 - d_0}{2} \right| \right) \delta. \end{aligned}$$

If we fix $|d_0|$ it is easy to show that the largest value $|(1 + d_0)/2| + |(1 - d_0)/2|$ takes, occurs when $d_0 = \pm \mathrm{i}|d_0|$ (where $\mathrm{i} = \sqrt{-1}$). This implies

$$|T(a) - T_0(a)| \leqslant \sqrt{1 + |d_0|^2}\,\delta. \qquad \square$$

Let us apply the lemma with $a_i = T_0^{-1}(c_i)$, $i = 1, 2$. Set $\alpha = \frac{1}{2}\sqrt{1/\gamma^2 - 1}$. Then if a is such that $|T_0(a) - c_0| \leqslant \alpha r_0$ we have $|a - a_0| \leqslant 2\alpha |a_1 - a_0|$ and so the lemma gives

$$|T(a) - T_0(a)| \leqslant \sqrt{1 + 4\alpha^2}\gamma t = t.$$

Finally, note that if λ and ρ are such that $\lambda^2/(\rho(1 - \lambda^2) + \lambda^2) < 1/\sqrt{5}$ then $\alpha > 1$. For instance if $\rho = 0.9$ then we need $\lambda < 0.649$, or if $\rho = 0.6$ we need $\lambda < 0.571$.

5. Implementation

The algorithm described above was implemented in the C programming language on a 64-bit processor Sun E450 running at 440 MHz and with sufficient RAM to avoid disk swapping. We did a case study on an actual fingerprint and tested the program on many randomly generated data sets. We use the least squares method for finding a global match from a correct local match. As already mentioned the least squares method should work better on smaller n, and is probably better than averaging in general (but harder to analyze). After all, our averaging method does not even use all the matching points that are available.

5.1. Case study

To test our algorithm on a real world situation, we applied it to a fingerprint recognition problem. We took two fingerprint images from the same person and with the feature points already extracted (see Fig. 7). This gives two point sets, each with 40 points, that we tried to match using our implementation. The program easily found a match involving 32 points. If we relax the matching distance up to 4 more points can be matched. In Fig. 8 we give a representation of the match found. The “x” marks represent the points

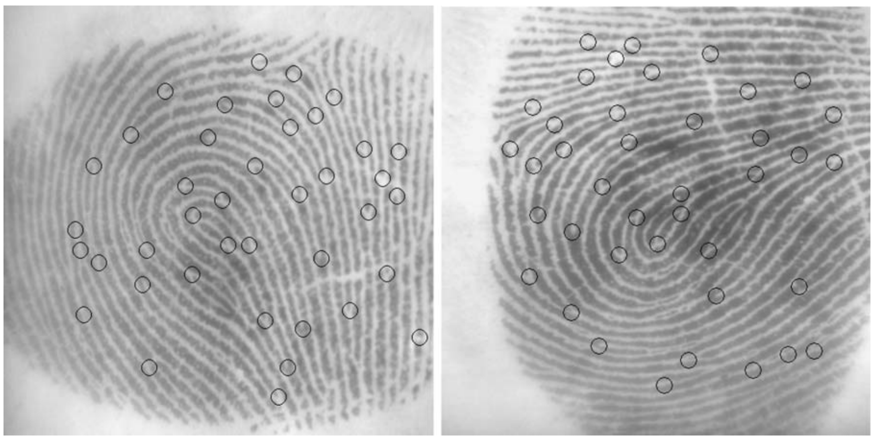
Fig. 7. Two different fingerprints from the same finger with the feature points circled.

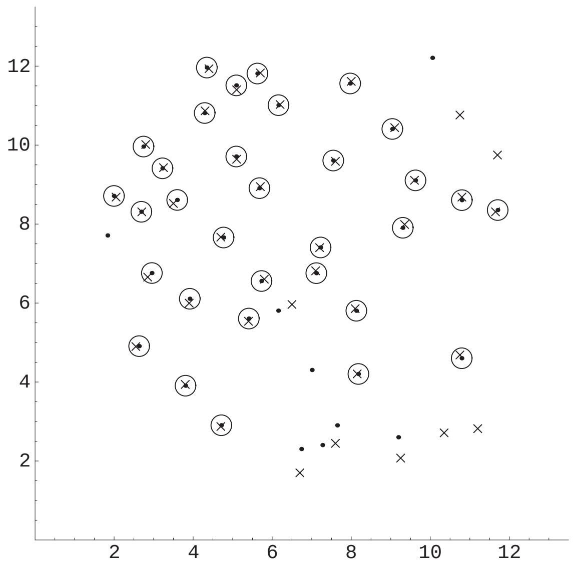

Fig. 8. The match our implementation found between the two sets of fingerprint feature points.

in the set S. The dots represent the images of the points in the M set (under the similarity transformation found by the program). Each point that was matched is circled with a circle of radius t, where t is the matching distance that was given to the program.

5.2. Random point sets

We also tested the program on a large number of randomly generated point sets. Some of the results are reported in Table 2.

For these experiments, in order to make them more realistic, we used normally distributed noise instead of uniformly distributed noise.

We always took the scene to be n points in the square $[0,1]\times[0,1]$. As a circle of radius $\sqrt{1/\pi}$ has the same area as this square, we took r (as in Eq. (1)) to be $\sqrt{1/\pi}$. So in the discussion below if we refer to λ it means, by Eq. (1), that $t=0.2821\lambda/\sqrt{n}$. We use a similar notation for the size of the noise. We will denote the amount of noise used by η. A given value of η corresponds to normal noise with standard deviation of $0.2821\eta/\sqrt{n}$.

Recall that for normal 2-D noise with standard deviation σ the proportion of points that will lie within $z\sigma$ of the mean is given by

$$1-\mathrm{e}^{-z^2/2}.$$

For instance, in order to find 95% of points one needs to go to 2.448 times the standard deviation.

The scenes were generated as follows. First we picked a random square of side length $\sqrt{m/n}$ in the $[0,1]\times[0,1]$ square. This is where the model will be. We chose m uniformly distributed points in the "model square" and a further $n-m$ uniformly distributed points outside the model square (but in $[0,1]\times[0,1]$). These n points, after a random re-ordering make up the scene.

The model was then generated by selecting ρm points from the model square, adding normal noise to these points, adding another $(1-\rho)m$ points (uniformly distributed in the model square) and then applying a random similarity transform to these m points.

We tested our implementation with various combinations of values for m, ρ and η. For each such choice we used scenes of size $2m, 4m$ and $8m$. For these combinations we tried to find those values of η, k, k_2, k_3 and k_0 (the meaning

Table 2
For each set of arguments, the best parameters and the average results over 100 trials

Arguments			Parameters					Average time in seconds for		
m	ρ	η	λ	k	k_2	k_3	k_0	$n=2$ m	$n=4$ m	$n=8$ m
50	0.95	0.175	0.428	5	1	2	3	0.02	0.053	0.089
100	0.95	0.175	0.428	6	1	2	4	0.051	0.099	0.23
200	0.95	0.175	0.428	6	1	2	4	0.13	0.257	0.518
400	0.95	0.175	0.428	7	1	2	5	0.347	0.704	1.401
800	0.95	0.175	0.428	8	1	2	5	0.759	1.596	3.802
50	0.6	0.25	0.61	9	4	4	4	0.504	1.065	2.852
100	0.6	0.25	0.61	10	2	4	5	1.208	2.102	4.479
200	0.6	0.25	0.61	10	2	3	5	2.854	6.612	11.853
400	0.6	0.25	0.61	11	3	4	6	8.257	17.007	34.475
50	0.9	0.4	0.84	8	2	2	5	0.138	0.322	0.613
100	0.9	0.4	0.84	8	1	4	6	0.282	0.607	1.614
200	0.9	0.4	0.84	9	1	3	7	0.797	1.495	3.61
400	0.9	0.4	0.84	9	1	5	7	3.001	5.862	11.491

Here m is the number of model points, n the number of scene points, ρ is the matching probability and η is the noise factor. λ, k, k_2, k_3 and k_0 are the best parameters found for each set of arguments.

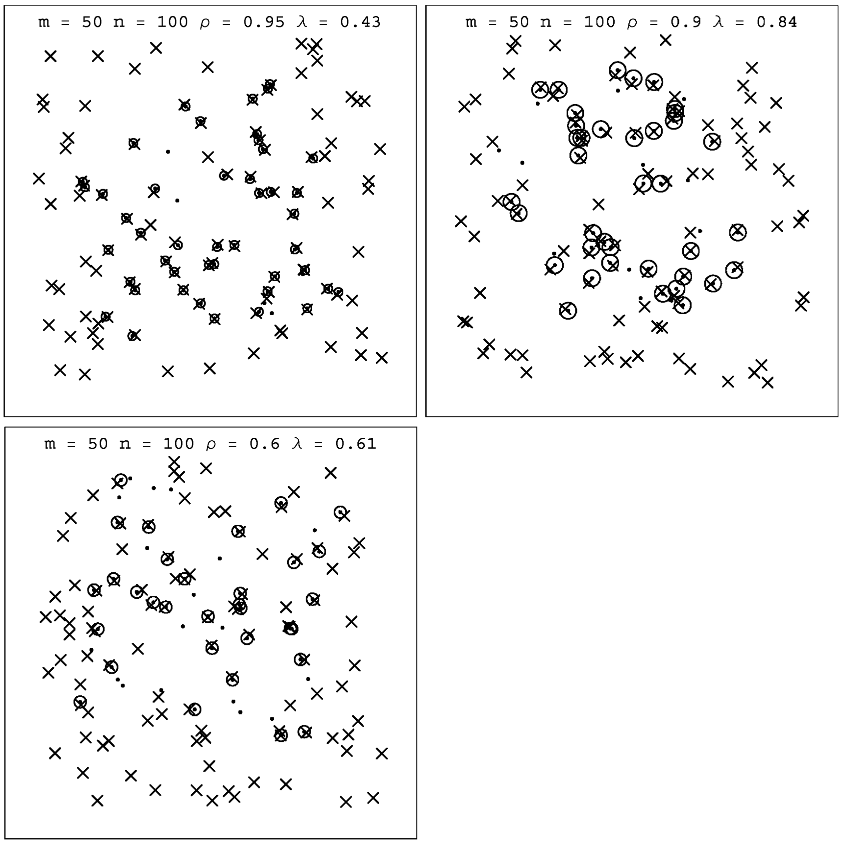

Fig. 9. Examples of matches found for different values of ρ and λ. The crosses are the scene points, the dots are the model points. Matching points are circled with a circle of radius equal to the matching distance.

of k_0 will be explained shortly) that gave a 100% success rate (out of 100 trials) and minimized the average time to find the match. Note that our analysis of the algorithm implies that these values do not depend on the scene size, only on m, ρ and η. So the values for λ and the k's reported in Table 2 are close to optimal for the given m, ρ and η. In the algorithm as described above, we declare a local match if $\rho(k-1)$ of the nearest neighbors match under a certain similarity transformation (see Section 3.3). When k is small we might want to modify this slightly, and so, for these tests, we declared a local match if k_0 or more nearest neighbors matched. In this way the running times can sometimes be improved over just choosing $k_0 = [\rho(k-1)]$.

For each 100 runs we report the average CPU time needed by the matching part of the algorithm (the precomputation time excluded, this turns out to be small compared to the matching time anyway).

As can be seen from the table (and a little computation) the best values for k, k_2 and k_3 are close to those predicted by the theoretical arguments in Section 4, i.e.,

$$k \approx \frac{\ln m}{2(\rho - \lambda^2/4)^2},$$

$$k_2 \approx \frac{\log 0.05}{\log(1-\rho)},$$

$$k_3 \approx \frac{2\lambda\sqrt{k}}{\sqrt{\pi}}.$$

In the table we present some of the extremes that our algorithm can handle. The first five entries is the ideal situation where η is small, meaning that the positions of the points are relatively accurate and ρ is high: in this case 95% of points were constructed to match. It should be noted though, that because we are using normal noise the proportion of points that actually match within t will always be less that ρ for any choice of t. For the choice of $\lambda = 0.428 = 2.45\eta$ we will have only approximately 95% of points that were constructed to match actually match within t. So in the end only about 90% of points will match.

The next 4 entries represent cases where only 60% of points are constructed to match and the error bound is relatively big. Even in these cases the algorithm succeeds in finding matches. This value, $\rho = 0.6$, is about as low as the algorithm will tolerate. For lower values of ρ the success rate of the algorithm starts dropping below 100%.

The last four entries represent the case where the error bound is now very big: the best λ is 84% of the average distance to the nearest neighbor. Again the algorithm succeeds in finding the match (note that ρ is relatively large though).

The behavior of the algorithm for intermediate values for ρ and λ can be extrapolated from the table.

See Fig. 9 for examples of what each of the three types of matches for 50 model points and a 100 scene points reported in Table 2, look like.

5.3. Non uniformly distributed data sets

As mentioned before, our algorithm would not perform well if the point sets are highly self-similar (e.g. a grid). If there was a lot of self-similarity we would get many incorrect local matches and this would significantly slow the algorithm down. In Ref. [5] the authors define a measure of self-similarity and it might be possible to use this to analyze our algorithm without having to assume that the points are uniformly distributed in M and S.

6. Conclusion

The design and analysis of data structures and algorithms is an important area of point pattern matching algorithms. More importantly, analyzing point set pattern matching is an integral component of pattern recognition problems.

The intent of this paper is the design and analysis of a probabilistic similarity transformation matching algorithm.

If m is the number of points in the model and n is the number of points in the scene, we give a $O(n(\log m)^{3/2})$ expected time algorithm for the point pattern matching problem and show that it is faster than any existing algorithms in the literature. We then describe some experimental results on both fingerprints and randomly generated data for the validation of our theoretical analysis. These results show significant improvements in running time. We prove our running time bound rigorously, but we also give a practical version of the algorithm and show that it performs well on real data sets.

Our experimental results show that our algorithm is applicable to a wide variety of problems. The algorithm performs well even if the allowed error is bigger that 80% of the average shortest distance to the nearest neighbor or the number of missing/extra points is high: even with only 60% of points matching the algorithm will succeed. Although we have concentrated on uniformly distributed point sets, there is good reason to believe that the algorithm will also work on data sets with outliers or clustering (but not on highly self-similar point sets).

References

[1] S. Ranade, A. Rosenfeld, Point pattern matching by relaxation, Pattern Recognition 12 (1980) 269–275.

[2] H. Ogawa, Labeled point pattern matching by fuzzy relaxation, Pattern Recognition 17 (5) (1984) 569–573.

[3] D.P. Huttenlocher, S. Ullman, Recognizing solid objects by alignment with an image, Internat. J. Comput. Vision 5 (2) (1990) 195–212.

[4] V.V. Vinod, S. Ghose, Point matching using asymmetric neural networks, Pattern Recognition 26 (8) (1993) 1207–1214.

[5] S. Irani, P. Raghavan, Combinatorial and experimental results for randomized point matching algorithms, Proceedings of the 12th Annual ACM Symposium on Computational Geometry, Philadelphia, PA, May 1996, pp. 68–77.

[6] S.-H. Chang, F.-H. Cheng, W.-H. Hsu, G.-Z. Wu, Fast algorithm for point pattern matching: invariant to translations, rotations and scale changes, Pattern Recognition 30 (2) (1997) 311–320.

[7] L. Boxer, Faster point set pattern matching in 3-D, Pattern Recognition Lett. 19 (1998) 1235–1240.

[8] S.-H. Chang, F.-H. Cheng, W.-H. Hsu, An $O(n^2)$ algorithm for 2-D point pattern matching, Pattern Recognition Lett., unpublished manuscript.

[9] H. Alt, L.J. Guibas, Discrete geometric shapes: matching, interpolation, and approximation, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier Science Publishers B.V., North-Holland, Amsterdam, 1999, pp. 121–153.

[10] A.G. Percus, O.C. Martin, Scaling universalities of kth-nearest neighbor distances on closed manifolds, Adv. Appl. Math. 21 (3) (1998) 424–436.

[11] M.T. Dickerson, D. Eppstein, Algorithms for proximity problems in higher dimensions, Comput. Geom. 5 (1996) 277–291.

[12] N.L. Johnson, S. Kotz, Distributions in Statistics: Discrete Distributions, Houghton Mifflin Co., Boston, MA, 1969.

[13] M. Okamoto, Some inequalities relating to the partial sum of binomial probabilities, Ann. Inst. Statist. Math. Tokyo 10 (1958) 29–35.

[14] W.E.L. Grimson, D.P. Huttenlocher, D.W Jacobs, A study of affine matching with bounded sensor error, Internat. J. Comput. Vision 13 (1) (1994) 7–32.

[15] P. Indyk, R. Motwani, S. Venkatasubramanian, Geometric matching under noise: combinatorial bounds and algorithms, Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland, USA, 1999, pp. 457–465.

[16] M.T. Dickerson, R.L. Drysdale, J.-R. Sack, Simple algorithms for enumerating inter-point distances and finding k nearest neighbors, Internat. J. Comput. Geom. Appl. 2 (3) (1992) 221–239.

About the Author—PAUL B. VAN WAMELEN received his Ph.D. degree in Mathematics from the University of California at San Diego in 1994. He then joined the faculty of the Department of Mathematics at Louisiana State University where he is currently an Associate Professor. He has had 2 year long visits to southern hemisphere universities, the University of South Africa in Pretoria (1997) and the University of Sydney in Australia (2002/03).

About the Author—ZI LI was a graduate student in the Department of Computer Science at Louisiana State University and graduated with a masters degree in System Science. He is currently working for Oracle in San Francisco.

About the Author—SITHARAMA S. IYENGAR is a Chairman and Distinguished Research Master Award winning Roy Paul Daniels Professor of the Computer Science Department at Louisiana State University. He has published extensively in the areas of high performance algorithms, data structures and sensor networks. He is a Fellow of ACM, AAAS and IEEE computer societies.